

# SPECTRE: A Fast and Scalable Cryptocurrency Protocol

Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar

School of Engineering and Computer Science,  
The Hebrew University of Jerusalem, Israel  
{yoni\_sompo,yoadlew,avivz}@cs.huji.ac.il

## Abstract

A growing body of research on Bitcoin and other permissionless cryptocurrencies that utilize Nakamoto’s blockchain has shown that they do not easily scale to process a high throughput of transactions, or to quickly approve individual transactions; blocks must be kept small, and their creation rates must be kept low in order to allow nodes to reach consensus securely. As of today, Bitcoin processes a mere 3-7 transactions per second, and transaction confirmation takes at least several minutes.

We present SPECTRE, a new protocol for the consensus core of crypto-currencies that remains secure even under high throughput and fast confirmation times. At any throughput, SPECTRE is resilient to attackers with up to 50% of the computational power (up until the limit defined by network congestion and bandwidth constraints). SPECTRE can operate at high block creation rates, which implies that its transactions confirm in mere seconds (limited mostly by the round-trip-time in the network).

Key to SPECTRE’s achievements is the fact that it satisfies weaker properties than classic consensus requires. In the conventional paradigm, the order between any two transactions must be decided and agreed upon by all non-corrupt nodes. In contrast, SPECTRE only satisfies this with respect to transactions performed by honest users. We observe that in the context of money, two conflicting payments that are published concurrently could only have been created by a dishonest user, hence we can afford to delay the acceptance of such transactions without harming the usability of the system. Our framework formalizes this weaker set of requirements for a crypto-currency’s distributed ledger. We then provide a formal proof that SPECTRE satisfies these requirements.

## 1 Introduction

Bitcoin is an open crypto-currency system that was invented and later deployed by Satoshi Nakamoto [14]. The main challenge in creating an open P2P system that

handles money is making the system resilient to Sybil attacks. In a Sybil attack, the attacker joins the system under multiple identities, and subverts its operation from within [4]. The main tool that Nakamoto used to overcome this problem is Proof-of-Work (PoW) – a cryptographic primitive that allows one machine to validate that another had invested some large amount of computational power [5]. While attackers can spawn additional identities, their constrained computational resources prevent them from solving too many PoW puzzles [10]. Nakamoto utilizes this primitive to maintain a consistent transaction ledger, called *the blockchain*. The blockchain is composed of individual *blocks*, which are essentially collections of transactions that are included in the ledger.

The Bitcoin system follows the replicated state machine approach [18]: nodes have local replicas of the ledger, and they reach agreement on the full order of blocks (and by extension on the full order of transactions) via the Nakamoto protocol. This allows nodes to agree on a consistent set of transactions by accepting only the first of any conflicting set.

Unfortunately, recent research has shown that the Nakamoto consensus has severe scalability limitations [3, 19, 7, 15]. If honest nodes do not synchronize fast enough, consensus cannot be achieved, and the irreversibility of transactions cannot be assured. To avoid this, Bitcoin was set to operate at extremely low rates. The protocol enforces a slow block creation rate, extending the blockchain only once every 10 minutes in expectation. Users must thus wait a long while to receive approval for their transfers.<sup>1</sup>

In addition to the restriction that this requirement places on the block creation rate, it also places a limit on the block size: blocks that are very large would take too long to propagate to other nodes. The combined limit on block creation rate and on block size effectively imposes a limit on the throughput of transactions that the system can process. If the throughput grows (either via an increase in block size or block creation rate), Nakamoto’s original guarantee no longer holds—attackers with less than 50% of the computational power can disrupt the system.

Disruptions to the consensus core of crypto-currency systems generally take one of two forms: attackers either compromise the *liveness* property of the system, which implies that they can ensure that transactions are not accepted into the system, or they compromise the *safety* property, which implies that transactions can be reversed and money can be double-spent. Indeed, as Bitcoin’s throughput is increased both attacks become feasible for attackers with less computational power; they can postpone the acceptance of *any* transaction *indefinitely*, and double spend any transaction they issue, regardless of the amount of time that has elapsed since its publication.

**Our contribution.** In this paper we present SPECTRE, a PoW-based protocol that can process a high throughput of transactions and maintain fast confirmation times while remaining secure from attackers with up to 50% of the computational power. SPECTRE relies on a data structure that generalizes Nakamoto’s blockchain into a direct acyclic graph (a block DAG). We provide extensive theoretical analysis of the

---

<sup>1</sup>Other cryptocurrencies that run slightly modified versions of Bitcoin’s code have sometimes opted for higher block creation rates. These come at some cost to their security

protocol, its security and scalability. We additionally provide simulation results that demonstrate SPECTRE’s advanced capabilities.

The name SPECTRE stands for “Serialization of Proof-of-work Events: Confirming Transactions via Recursive Elections”.

**Properties of SPECTRE.** SPECTRE’s design is intended to avoid the need for nodes to reconcile their different world views regarding the identity of a selected chain at the time of block creation. By maintaining a full DAG of blocks, SPECTRE can allow miners to create blocks concurrently and much more frequently. Agreement on the main chain is not required when a block is created,<sup>2</sup> nor is any knowledge of the propagation delay in the network required to run a mining node. Instead, the decision on which transactions have been approved by the system is made after the fact, by anyone that observes a recent version of the block DAG. SPECTRE guarantees that such decisions will be consistent and that transfers remain secure from double spending or delays.

The block creation rate in SPECTRE is not constrained by network propagation delays, implying that large and fast blocks can be created, up to the limit imposed by network congestion and bandwidth that is required from nodes to receive a full copy of all transactions.<sup>3</sup> This results in fast confirmation times of transactions, especially so for small sized attackers.<sup>4</sup> Below are some confirmation times that can be achieved with SPECTRE, by creating 10 blocks per second. These numbers assume that blocks (e.g., of size 100 KB) propagate to the vast majority of mining nodes within 5 seconds.

protocol	$\lambda$	$d$	$\alpha$	time until $\epsilon = 0.01$
SPECTRE	10	5	0.05	10.07 sec
			0.1	11.3 sec
Bitcoin	1/600	5	0.05	2700 sec
			0.1	4500 sec

In Section 6 we show that the acceptance times in SPECTRE can be further accelerated by tightening the analysis. We additionally show that confirmation times are highly affected by the delay diameter; we believe that 5 seconds for 100 KB blocks represents a sufficiently conservative estimate (for example, the “Bitcoin Relay Network” performs much better with 1 MB blocks).

An additional benefit of SPECTRE’s high block creation rate is that mining rewards have significantly lower variance compared to Bitcoin: if the block creation rate is set to 10 blocks per second then a solo-miner will receive rewards 6,000 times more often. This will reduce the need for large mining pools, and will therefore contribute to the system’s decentralization.

<sup>2</sup>In contrast, Bitcoin nodes must choose a single chain to extend when building blocks at the moment of block creation.

<sup>3</sup>We note that these limits are not too restrictive: 2,000 bitcoin transactions take up roughly 1MB, which implies that a 10Mbps connection could transmit 20,000 transactions per second, rivaling the global transaction rate of Visa at peak times.

<sup>4</sup>Merchants conducting transactions at a physical point of sale are mostly concerned with fast confirmation times while securing themselves against small attackers (hence their willingness to settle for 0-confirmation transactions in Bitcoin).

The resilience of SPECTRE to attackers with up to 50% of computational power comes at a cost. While Nakamoto’s protocol resolves conflicts between *any* pair of transactions (by deciding their order), SPECTRE only guarantees this with respect to *some* pairs of transactions. In SPECTRE, if two conflicting transactions were published at about the same time, the identity of the prevailing transaction might remain undetermined for arbitrarily long periods of time. Our key insight is that in the context of crypto-currencies transactions only conflict if they move the exact same funds to two different locations, which implies that they were both generated by the original owner of the funds (cryptographic signatures ensure that only the owner may move his funds). Hence, an honest participant will *never* create such conflicts, and we can afford to delay the decision regarding transactions that have been *visibly* double spent.

**The Requirements from Distributed Ledgers (RDL) framework.** To make our arguments precise we provide a formal framework to reason about the required properties of distributed ledger protocols for crypto-currencies, which is of independent interest. By specifying weaker properties that distributed ledgers must satisfy, we include solutions such as SPECTRE that provide better security and performance.

According to the RDL-framework, a distributed ledger protocol must provide a pair of procedures *GetAccepted* and *ChkRobustAccept*. The *GetAccepted* procedure takes as input a user’s current world view (in our case, this is in the form of the block DAG that it observes locally) and outputs a consistent subset of *accepted transactions*.<sup>5</sup> *ChkRobustAccept* enables the user to determine whether he can safely consider funds to have been transferred. For example, a merchant receiving payments will only deliver the good or service to the buyer once he had determined that the funds are securely his. *ChkRobustAccept* takes as input the user’s world view, a transaction from the accepted subset, and  $\epsilon$ ; it returns ACCEPT if the user can safely consider funds to have been received with probability of transaction reversal that is smaller than  $\epsilon$ , or WAIT otherwise. In the former case we say that the user has  $\epsilon$ -accepted the transaction. We require that the following properties be satisfied:

- **Consistency:** *GetAccepted* returns a consistent set.
- **Safety:** if an honest user  $\epsilon$ -accepted a transaction (using *ChkRobustAccept*) then w.h.p. eventually any honest user will do the same.
- **Progress:** if an honest user  $\epsilon$ -accepted a transaction then w.h.p. it will eventually  $\epsilon'$ -accept it (for any  $\epsilon'$ ).
- **Weak Liveness:** if a transaction is contained in a published block, then as long as its inputs remain  $\epsilon$ -accepted, and as long as no conflicting transactions are published, eventually an honest node will  $\epsilon$ -accept it.

We formalize these properties in Section 4.

The highlight of this work is the SPECTRE protocol, whose *GetAccepted* and *ChkRobustAccept* procedures satisfy these requirements:

---

<sup>5</sup>The consistency notion depends on the system of transactions that the ledger serves. In Bitcoin for example, two transactions are in conflict if they spend the same outputs.

**Theorem** (informal). *If honest nodes possess a majority of the computational power, then regardless of the network’s communication delay, SPECTRE satisfies Consistency, Safety, Progress, and Weak Liveness.*

**Fast acceptance times and scalability.** A scalable protocol is one in which waiting times for transaction processing remain short even under high transaction throughput. In SPECTRE, we achieve waiting times that are  $\mathcal{O}\left(\frac{\ln(1/\epsilon)}{\lambda(1-2\alpha)} + \frac{d}{1-2\alpha}\right)$ , where  $d$  is a bound on the time it takes to reach honest nodes that have at least  $1-\alpha$  fraction of the computation power (and  $\alpha$  is the fraction of computational power of the attacker along with additional distant nodes),  $\epsilon$  is a bound on the probability that nodes are willing to tolerate of transaction reversal, and  $\lambda$  is the rate of block creation. Considering the expression above, it is easy to see that SPECTRE operates best with low propagation delays, and high block creation rates. As block creation rates are increased, the first term becomes negligible, and the dominating term is  $\mathcal{O}(d/(1-2\alpha))$ , which is essentially a measure of the propagation delay to reach a large amount of honest nodes in the network.

**Simulations.** In addition to the theoretical analysis, we provide the results obtained by running the protocol on a simulated network topology. We show how waiting times change in practice as parameters are adjusted, and provide evidence for the protocol’s applicability.

**Organization of the paper.** The remainder of this paper is organized as follows: In the subsequent subsection we discuss related work. We describe the basic operation of SPECTRE in Section 2. Next, in Section 3, we provide examples of attacks and provide intuition for why they fail to disrupt the protocol. A formal model is provided in Section 4. We then return to present SPECTRE’s procedures formally in Section 5. In Section 6 we present our simulation results for the protocol and evaluate its performance. We discuss some implementation details in Section 7, including more details about minting, denial-of-service attacks, and re-targeting. In Section 8 we outline the proof of the main theorem that we have stated above. We conclude in Section 9. Finally, we provide the full proof in Section 10.

## 1.1 Related Work

Previous research has produced several suggestions for protocols that attempt to address the security-scalability challenge, but all protocols still provide a total order over blocks:

The GHOST protocol is an alternative chain selection rule that gradually chooses a tree of blocks until converging on a single chain [19]. GHOST utilizes pointers from off-chain blocks in an attempt to overcome the disadvantage of honest nodes compared to a centralized attacker. It can be shown that the Liveness property of GHOST can be attacked in several ways, as was demonstrated by [11].

The use of block DAGs was proposed in the Inclusive protocol [13], in which throughput was increased by combining discarded blocks into the ledger. However, the protocol modification of Inclusive [13] still relies internally on a simple chain-based protocol to provide a total order over blocks. As such, it mitigates but does not

avoid the security-scalability trade-off. The Inclusive paper further includes a game theoretic analysis of the incentives of nodes to embed different transactions in their blocks (without the ability to coordinate).<sup>6</sup>

Bitcoin-NG [6] provides a clever chain structure that is composed of two type of blocks key blocks that require PoW but contain no transactions and mini-blocks that do not require proof of work but do contain transactions. Indeed, Bitcoin-NG manages to obtain a significant scalability increase, but its key blocks are still generated slowly, hence, confirmation times remain high.

The work of Pass and Shi [16] constructs an interesting hybrid model that combines consensus protocols, building on a previous work by [12, 2]. In Hybrid Consensus, the Nakamoto consensus is used only to select a subset of nodes out of the entire (*a priori* unknown) set of nodes. The selected committee then communicates internally and runs some conventional consensus algorithm to confirm new transactions. Arguably, the consensus protocol run by Hybrid Consensus’s rotating committee has its own scalability limitations. Concretely, committee members must remain available online for some period after they are chosen, and need to directly communicate with one another, which introduces a new set of considerations, e.g., availability, DoS attacks, and anonymity concerns. In contrast, SPECTRE only relies on miner nodes to create blocks of transactions that extend their local DAGs, and to publish them. Miners in SPECTRE are not directly involved in any explicit consensus protocol and moreover can operate with little regard of other nodes’ synchronization status. Lastly, Hybrid Consensus is only resilient to attackers that control less than 33% of the total computational power, whereas SPECTRE is resilient to any attacker up to 50%. This gap is inevitable, as Hybrid Consensus satisfies stronger properties than SPECTRE and the RDL-framework set out to meet: It achieves regular (rather than weak) Liveness and ensures a full order of transactions.<sup>7</sup>

To the best of our knowledge, this is the first work to propose a *native* PoW-based protocol for crypto-currencies that does not rely on the selection of a main chain.

## 2 The SPECTRE Protocol (informal)

In this section we describe the basic operation of SPECTRE, via informal instructions. A formal specification of the procedures will appear in Section 5.

### 2.1 Basic notation

A block DAG is typically denoted by  $G = (C, E)$ , where  $C$  represents blocks  $E$  represents the hash references. For a block  $z \in C$  we will frequently abuse notation and write  $z \in G$ .  $past(z, G) \subset C$  denotes the subset of blocks reachable from  $z$ , and similarly  $future(z, G) \subset C$  denotes the subset of blocks from which  $z$  is reachable (these are blocks that are provably created before and after  $z$ , correspondingly). Note that an edge in the DAG points back in time, from the new block to previously created

---

<sup>6</sup>We build on this argument, and indeed assume that nodes will maximize their profits by avoiding transaction “collisions” and will try to embed unique content in their blocks.

<sup>7</sup>The 33% is tight when aiming at classic consensus properties [16].

blocks which it extends. We denote by *anticone* ( $z, G$ ) the set of blocks that the DAG does not directly order compared to  $z$ , that is:  $G \setminus (\text{past}(z, G) \cup \{z\} \cup \text{future}(z, G))$ . The unique block *genesis* is the block created at the inception of the system, and every valid block must have it in its past set. The set *past* ( $b, G$ ), in sharp contrast to *future* ( $z, G$ ) and *anticone* ( $z, G$ ), is fixed once and for all at the creation of  $b$ , and does not require specifying the “context”  $G$ . Finally, we relate to a hypothetical block, *virtual* ( $G$ ). This block satisfies  $\text{past}(\text{virtual}(G)) = G$ . While its role is merely methodological, *virtual* ( $G$ ) can also be thought of as representing the next block that a node whose current observed DAG is  $G$  attempts to create.

## 2.2 The generation of the block DAG

We begin by reiterating the extremely simple instructions of SPECTRE to miners, described in Section 4:

1. *When creating or receiving a block, transmit the block to all peers.*
2. *When creating a block, embed in its header a list containing the hash of all leaf-blocks in the locally-observed DAG.*

*In particular, all blocks are incorporated into the DAG.* Note that these instructions allow miners to operate concurrently irrespective of potential conflicts in the contents of their blocks. The

## 2.3 Layout

SPECTRE is composed of two layers: interpreting the block DAG, and estimating the robustness of this interpretation. Its layout can be summarized as follows:

1. **GetAccepted**( $G$ ):
  - (a) Calculate pairwise relation  $\prec$  on blocks in  $G$  (Alg. 1)
  - (b) Accept transactions that defeat all conflicts (Alg. 2)
2. **ChkRobustAccept**( $G, tx, \epsilon$ ):
  - (a) Estimate robustness of the relation  $\prec$  for relevant pairs of blocks (Algs. 3, 7)
  - (b)  $\epsilon$ -accept  $tx$  if the block that contains it robustly precedes all counterparts (Algs. 4,5,6)

The first layer is an implementation of *GetAccepted*. Given a block DAG, this procedure outputs a consistent subset of its transactions. This is done by first computing a pairwise relation  $\prec$  over the blocks (in the context of some  $G$ , we informally say that block  $x$  *precedes* or *defeats* block  $y$  if  $x \prec y$ ). Then, any transaction is accepted if its blocks defeats the blocks of all conflicting transactions in its anticone. The relation  $\prec$  is generated by a pairwise vote procedure that occurs independently for every pair of blocks.

Although we may at times refer to  $\prec$  as though it orders blocks, we stress that  $\prec$  *is not necessarily a transitive relation*. It is possible to have a series of blocks that defeat each other cyclically.<sup>8</sup> This is in fact the way SPECTRE utilizes the weaker requirements in the RDL-framework (the lack of total ordering over blocks).

The next layer implements *ChkRobustAccept*. Here, similarly, we first estimate the robustness of  $\prec$  regarding certain pairs of blocks, namely, the one containing the given transaction vs blocks containing possible conflicts; then we  $\epsilon$ -accept the given transaction if its containing block indeed defeats all of these blocks with a sufficient margin of safety.

## 2.4 Pairwise ordering of blocks

The basic layer of SPECTRE involves deciding on a pairwise order over the block DAG, which we denote  $\prec$  informally. Fix two blocks  $x, y \in G$ . In order to decide if  $x \prec y$  or  $y \prec x$ , we interpret the structure of the DAG as representing an abstract vote. Every block  $z \in G$  is considered a voter with respect to the pair  $(x, y)$ , and *its vote is inferred from its location in the DAG*. We represent a vote by a number in  $\{-1, 0, +1\}$ , and we denote its voting-profile on all pairs by  $vote(z, G)$ .  $vote_{x,y}(z, G) = -1$  represents  $x$  preceding  $y$  ( $x \prec y$ ),  $vote_{x,y}(z, G) = +1$  represents  $y$  preceding  $x$ , and  $vote_{x,y}(z, G) = 0$  represents a tie. Importantly,  $vote(z, G)$  is an asymmetric relation:  $vote_{y,x}(z, G) = -vote_{x,y}(z, G)$ .

To simplify presentation, we associate a vote with  $virtual(G)$  as well. Recall that the virtual block of  $G$  is a hypothetical block which satisfies  $past(virtual(G)) = past(G)$  (Section 4). The vote of  $virtual(G)$  represents essentially the aggregated vote of the entire block DAG. The basic rules of  $z$ 's vote, for any  $z \in G \cup \{virtual(G)\}$ , are as follows:

- if  $z$  is in  $future(x)$  but not in  $future(y)$  then it will vote in favour of  $x$  (i.e., for  $x \prec y$ ).
- if  $z$  is in  $future(x) \cap future(y)$  then  $z$ 's vote will be determined recursively according to the DAG that is reduced to its past. If the result of this vote is a tie,  $z$  breaks it arbitrarily.<sup>9</sup>
- if  $z$  is not in the future of either blocks then it will vote the same way as the vote of the majority of blocks in its own future.
- if  $z$  is the virtual block of  $G$  then it will vote the same way as the vote of the majority of blocks in  $G$ .

Additionally,  $z$  will vote for itself to succeed any block in  $past(z)$  and to precede any block outside  $past(z)$ .

Figure 1 illustrates the voting procedure with regards to a single pair of blocks  $(x, y)$ . Additional examples along with intuition regarding this key algorithm are provided in Section 3.

<sup>8</sup>This is related to the Condorcet paradox in social choice [1].

<sup>9</sup>We can use information encoded in  $z$ 's header, e.g., explicit instructions for tie-breaking, or using the lexicographical ordering of (hashes of) tied blocks, etc.



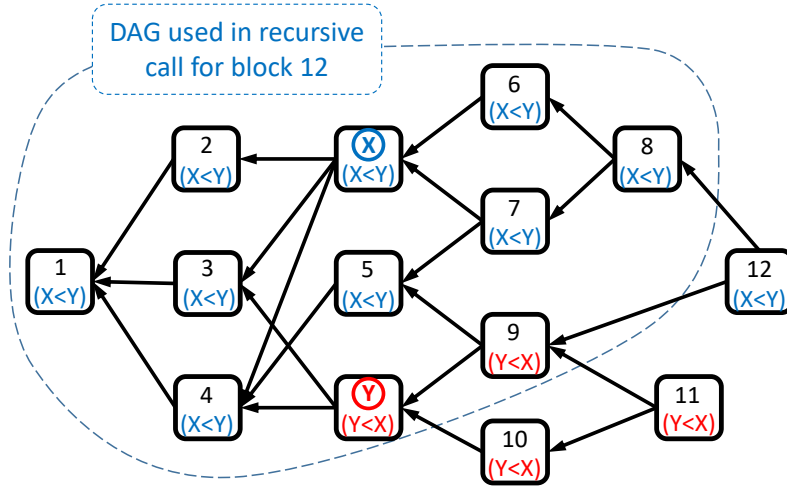


Figure 1: An example of the voting procedure on a simple DAG. Blocks  $x$  and 6-8 vote  $x \prec y$  as they only see  $x$  in their past, and not  $y$ . Similarly, blocks  $y$  and 9-11 vote  $y \prec x$ . Any block from 1-5 votes  $x \prec y$ , because it sees more  $x \prec y$  voters in its future than  $y \prec x$  voters. Block 12 votes according to a recursive call on the DAG that does not contain blocks 10,11,12.

It is easy to see that all votes respect the DAG’s topology: If  $x$  is reachable from  $y$  then all blocks vote unanimously  $x \prec y$ . Observe further that the vote of any  $z$  that has either  $x$  or  $y$  in  $\text{past}(z)$  is fully determined by  $\text{past}(z, G)$ , a set which is fixed at  $z$ ’s creation. Accordingly, we refer to such a  $z$  as a *strong voter* w.r.t. the pair  $x$  and  $y$ . In contrast, other  $z$ ’s are called *weak voters*, as their vote can be changed by future events: their votes depend on votes of blocks in their future sets (which grow over time).

The pairwise ordering of SPECTRE has the following highly valuable property: *Once a block is published, the set of blocks that precede it in the pairwise ordering closes fast—w.h.p. it only consists of blocks that were published before or right after its publication.*

The implications of this guarantee to the security of transactions is immediate, at least at the intuitive level: A user whose transaction is embedded in some published block  $x$  can guarantee its safety by waiting some time after  $x$ ’s publication before accepting it; he is then guaranteed that any block published later on – and that might contain a conflicting transaction – will be defeated by  $x$  hence will not threaten the acceptance of his transaction. In Section 3 we will explain how this guarantee is achieved.

## 2.5 Accepting transactions

Equipped with the pairwise relation over blocks, we now turn to construct the set of accepted transactions. Basically, we mark a transaction as accepted iff all three

conditions below hold true:

- all of its inputs have been accepted.
- all conflicting transactions that are in its anticone set (i.e., that are not related to it topologically) are contained in blocks that are defeated by the block containing the transaction.
- all conflicting transactions that are in its past set (i.e., that precede it in the DAG, topologically) have been rejected.

### 3 Intuition and Examples

In this section we provide some basic explanations and intuitions regarding the operation of SPECTRE. We focus primarily on explaining the ideas underlying Algorithm 1 that is at the core of the protocol. We later go on to present examples for simple attacks that shed some light on how resilience is achieved.

**Intuition 1** (Vote in favour of visible blocks). If a block  $x$  is known by honest participants, their blocks will include it in their past. Given that blocks vote in favour of blocks in their past (over other unknown blocks), and given that honest nodes publish their blocks quickly, hidden attacker blocks lose votes.

**Intuition 2** (Majority amplification). Given blocks  $x, y$  that contain potential conflicts, blocks that are generated by honest participants after their publication reference both of them in the DAG. According to Algorithm 1, these new blocks adopt the vote of the sub-DAG in their past with regards to  $x$  and  $y$ . Thus, if block  $x$  defeats block  $y$ , additional votes that support this decision are added, and the attacker will find it more difficult to reverse the vote.

**Intuition 3** (Referencing recent blocks is beneficial). Blocks from the past vote according to their future. Thus if an attacker creates a block that does not reference recent blocks, it is at a disadvantage compared to other blocks that do (it loses votes from recent blocks it did not reference and did not “convince”).

**Intuition 4** (Votes from the past counter pre-mining attacks). Consider an attacker that creates a block  $y$ , withholds it, and constructs many blocks on top of it over an extended period of time. After a long while, a conflicting transaction is released to the network, and eventually ends up in some block  $x$ . Block  $y$  has many blocks (built by the attacker) that reference it. Thus, if only votes from the future are counted, block  $y$  would prevail even if  $x$  is allowed to accumulate some votes. In SPECTRE, blocks that were created by honest nodes while  $y$  was withheld, look to their future for their votes. These will usually vote in favour of  $x$  and will usually outnumber the attacker blocks that were created when  $y$  was withheld (an example of pre-mining appears in Figure 3).

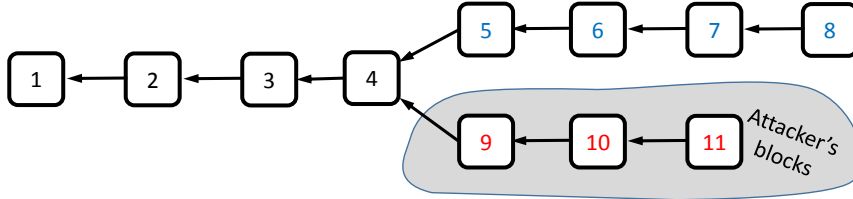


Figure 2: SPECTRE coincides with the longest-chain rule when it is applied to “simple” chains of blocks. In the depicted DAG, the chain ending at block 8 is longer and would be selected in the longest chain protocol. In SPECTRE each one of the blocks 5,6,7,8 defeats each of the blocks in 9,10,11. Consider for instance blocks 6 and 10 and the pairwise vote that involves them. Blocks 6-8 vote strongly  $6 \prec 10$ , as they see block 6 in their past but not block 10. Block 5 is a weak voter, as it sees neither 6 nor 10 in its past, hence it votes as the majority of its future (thus voting  $6 \prec 10$  as well). For similar reasons, blocks 9-11 all vote  $10 \prec 6$ . Block 4, at the fork of the two chains, is a weak voters as well, as it sees neither 6 nor 10 in its past; it therefore votes according to the majority of future blocks. As block 4 sees four votes in favour of  $6 \prec 10$ , and three votes in favour of  $10 \prec 6$ , it will vote in favour of  $6 \prec 10$ . Blocks 1-3 similarly vote according to their future, and see an increasing number of votes for  $6 \prec 10$ , adding their own vote to the result. Thus, the end result is that 6 defeats 10.

### 3.1 Equivalence to longest-chain

We first demonstrate how SPECTRE coincides with Bitcoin’s longest-chain rule, in the case of a “simple” fork between two chains. Consider the DAG illustrated in Fig. 2. In Bitcoin, the longer chain would be selected. Similarly, in the pairwise ordering of SPECTRE, each of the blocks in the longest chain 5,6,7,8 would defeat each of the blocks in the shorter one 9,10,11. To see why this is true refer to the caption of the figure.

We now turn to examine two different attack scenarios, which we name *double-spending*, and *censorship*. Recall the requirement from our miner protocol: each miner is required to (i) reference recent blocks, and to (ii) publish his blocks immediately. Each attack is basically a violation of one of these requirements. In the double-spending attack, the attacker delays the publication of a set of blocks (that includes a conflicting transaction), and in the censorship attack he publishes blocks but “ignores” a certain block and transactions inside it, hoping to convince nodes that it did not secure enough votes, and thus cannot be accepted.

### 3.2 Example of a double-spending attack

Figure 3 depicts an (unsuccessful) double-spending attack. The attack is composed of three main phases:

**Phase I: Pre-mining.** In phase I, the attacker begins building blocks and

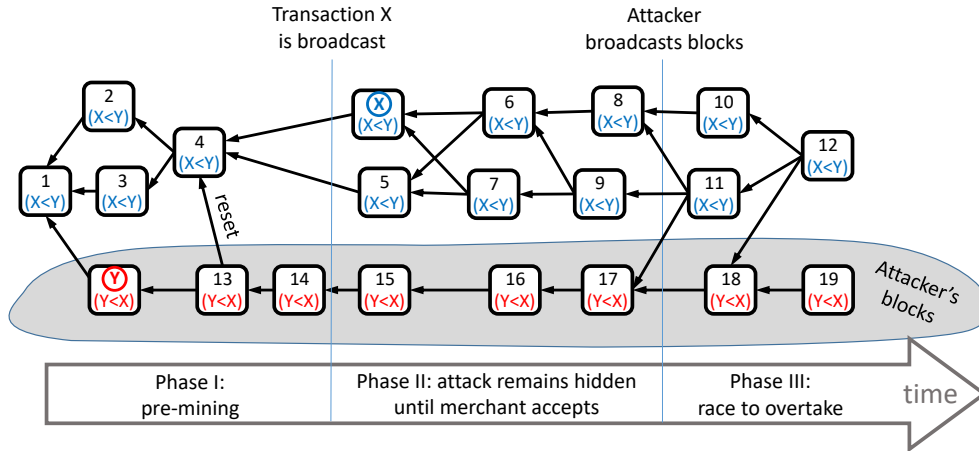


Figure 3: An example of the voting procedure on a DAG in which a double spending attack is (unsuccessfully) attempted. Blocks  $x$  and 6-8 vote strongly  $x \prec y$  as they only see  $x$  in their past, and not  $y$ . Similarly, blocks  $y$  and 9-11 vote strongly  $y \prec x$ . Blocks 1-5 vote  $x \prec y$ . This is because they see more  $x \prec y$  voters in their future than  $y \prec x$  voters. Block 12 votes according to a recursive call on the DAG that does not contain blocks 12,19.

withholding them from the network. The first block that is constructed (named block  $y$ ) contains a transaction that will later conflict with the transaction sent to the honest nodes. Blocks built by the attacker ideally form a chain, and due to the voting rules in SPECTRE, will all vote  $y \prec x$  (blocks  $y, 13, 14$ ). Blocks built by the honest node are unaware of  $y$  (and also of  $x$  that is yet to be created), and will eventually vote according to the majority of future votes. During this phase, attacker blocks reference honest blocks that are built (in hopes of later convincing them to vote  $y \prec x$ ). After some time, the attacker transmits the transaction to the network, and proceeds to phase II.

Notice that at the exact time that phase I ends, the attacker has more blocks above block 4 than honest nodes have, so it starts at an advantage: it will more easily sway the vote of block 4 towards  $y \prec x$  (this advantage later disappears as honest nodes typically build blocks faster than the attacker).

**Phase II: Waiting for acceptance.** The attacker now continues to build blocks in secret. If he publishes his blocks, then his conflicting transaction will be visible to all, and the double-spend will be detected. Instead, he waits for block  $x$  to receive sufficient weight (in the form of blocks built on top of it) so that the recipient of the transaction in  $x$  accepts it, and provides the attacker with some service or product. During this phase, attacker blocks that are created (blocks 15-17) vote  $y \prec x$ , as the attacker is careful to have them reference only his secret chain, and never indirectly reference block  $x$ . Honest blocks created during this phase will typically vote  $x \prec y$  since  $y$  is hidden from them. Some small number of blocks (created before  $x$ , and propagated to the whole network – block 5 in this example) do not reference  $x$ , and

so will vote according to the result of future votes.

**Phase III: Race to overtake.** Once  $x$  was  $\epsilon$ -accepted by the victim, the attacker wishes to publish his secret blocks in hopes of causing his conflicting transaction in  $y$  to precede  $x$ . In this case, the transaction in  $x$  will be considered rejected, and the payment will be canceled (leaving the attacker with an item he did not pay for). He publishes his secret chain (which from this point on is referenced by honest nodes), and continues to build upon it. Blocks that he builds, again do not reference  $x$ , and so they vote  $y \prec x$ , supporting his goal. New honest nodes are for the first time exposed to the conflicting transaction  $y$ , and thus vote according to the result in the sub-DAG in their past.

**Why the attack fails.** First, notice that the attacker in the above example creates fewer blocks in each phase than the honest nodes. This will usually be the case if attackers have less computational power than all honest nodes. “Poisson bursts” in block creation by the attacker are possible, and this will allow him to overtake the network, but these are less likely if the attack lasts for a long period of time. The defender can control the length of phase II by waiting a long while before accepting the transaction, which decreases the probability of such bursts. If phase II is long enough,  $x$  will have more votes in this period than  $y$ . Weak blocks in the past of  $x$  will then vote in favour of  $x$ , according to this majority. Such blocks that look at their future begin a cascade: each block further in the past adds a vote that agrees with the majority of future blocks and thus strengthens the decision. The greater the majority obtained in Phase II, the less likely it is that the attacker will be able to catch up from behind in Phase III. The attack therefore depends heavily on successfully swaying the votes of blocks that were created just before  $x$  (e.g., block 4).

It is important to note that an attacker that creates more blocks in expectation than the honest network will succeed in carrying out this attack. The blocks voting  $y \prec x$  would outnumber those who vote to the contrary. Hence the 50% threshold in Theorem 1.

### 3.3 Example of a censorship attack

Figure 4 depicts an (unsuccessful) censorship attack. The attack is composed of a single main phase during which an attacker creates his own blocks, publishes them instantly, but also ignores (and does not reference) recent blocks created by the honest network. The figure depicts (in stage I on the left side) the current state of the blockchain (where all blocks are published at this point). An honest participant that then observes the network and wishes to tell if a transaction in block  $x$  is secure, can see a large number of blocks that do not reference  $x$ . These blocks are not guaranteed to vote in favour of  $x$ . An attacker may later insert a conflicting transaction  $y$  and add blocks atop it (this projected attack is depicted on the right-hand side of the figure). These may potentially sway previously created attacker blocks to vote against  $x$ .

The main risk from the censorship attack is that merchants, upon seeing the attacker’s blocks, will consider transactions in block  $x$  not sufficiently secure. This could potentially delay the acceptance of transactions forever. Our analysis of SPECTRE shows that even in this case the merchants accept transactions quickly

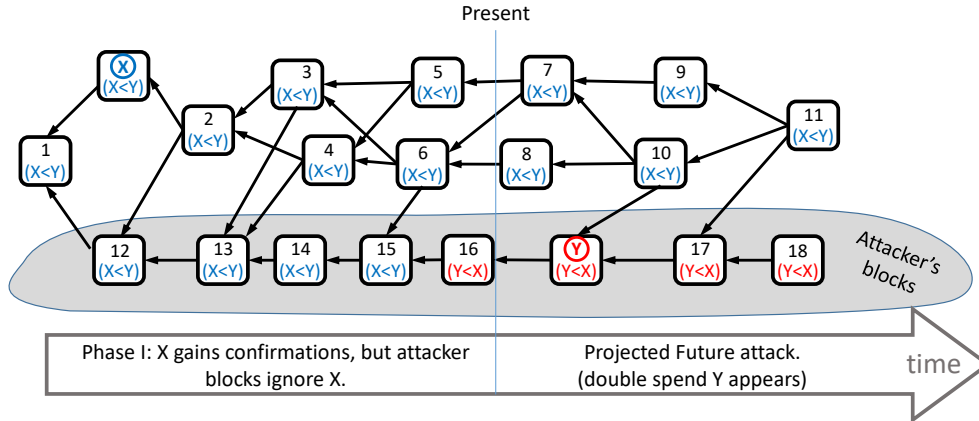


Figure 4: An example of the voting procedure on DAG in which an unsuccessful censorship attack is depicted. The left side depicts the current state of the block DAG. The right-hand side depicts its likely future development. Blocks 12-16 do not add strong votes to  $x$ . Can they be convinced to vote for block  $y$  when it appears? Will they further sway other blocks in their past? The vote of each block in this projected future are depicted: Blocks 2-9 vote strongly for  $x$  as they see it in their past (but not  $y$ ). Blocks 17-18 similarly vote strongly for  $y$ . Block 16 is indeed convinced to vote for  $y$  as more blocks in its future vote for  $y$  than for  $x$ . Blocks 1, 12-15 vote for  $x$ . They each see more votes in favour of  $x$  than votes in favour of  $y$  in their future. Blocks 10-11 see more  $x < y$  voters in their past when they make a recursive call.

(and securely).

## 4 Model and Notation

Below we provide the model for this paper. The familiar reader will notice that we share most of the primitives and features of the Nakamoto consensus and Bitcoin, e.g., PoW, blocks of transactions, etc.

**Miners and network.** The system is operated by nodes (also called *miners*), connected by a P2P network  $\mathcal{N}$ . We denote by  $honest \subset \mathcal{N}$  the set of nodes that always follow the protocol (in a manner that will become precise later). We assume that the honest nodes make up a connected component in the network graph, and that messages that they send are forwarded to their peers via gossip algorithms (networking attacks that disrupt or slow such connectivity were considered in Bitcoin before [9, 8]).

**Transactions.** Users create, sign, and publish transactions. As several copies of the very same transaction may appear in the system, we denote by  $[tx]$  the equivalence class of  $tx$ , containing all of the different copies of the same transactions. The consistency of a transaction  $tx$  is governed by two relations: First, the inputs of a transaction denoted  $inputs(tx)$  are the (equivalence classes of) transactions that must be accepted before  $tx$  can be accepted (these are essentially the transactions

that have provided the money that is being spent in  $tx$ ).

Second, two transactions  $tx_1$  and  $tx_2$  that are not equivalent may be in conflict, i.e., they cannot be accepted by the system at the same time (this is if they are conflicting orders to move the same money to different destinations). We denote by  $\text{conflict}(tx_1)$  the set of transactions that conflict transaction  $tx_1$  (this is a symmetric relation, and we assume that copies of  $tx_1$  do not mutually conflict).

**Blocks and the block DAG.** We denote the creator of block  $b$  by  $\text{node}(b)$  and the time it was created by  $\text{time}(b)$ . Every block obtains a unique id (in the form of the cryptographic hash of its header). The body of a block contains a subset of transactions (that were already created by  $\text{time}(b)$ ), chosen freely by  $\text{node}(b)$ .<sup>10</sup> The notation  $Z_G(tx)$  stands for all blocks in  $G$  that contain  $tx$ . We slightly abuse notation and denote  $tx \in G$  where it should read  $\exists b : tx \in b \in G$ . The block size is limited by some constant  $B$ . The block header contains an ordered list of references to other blocks (that were already created by  $\text{time}(b)$ ), by simply listing their ID (a cryptographic hash). As a result, blocks are formed in the structure of a DAG—a Directed Acyclic Graph whose edges correspond to the references between blocks (in contrast, the Nakamoto consensus utilizes a block chain with only one pointer to a single predecessor).

A block DAG is typically denoted by  $G = (C, E)$ , where the vertex-set  $C$  represents the blocks and the edge-set  $E$  represents the hash references. The DAG is essentially a representation of the causal order between block creation events. For any  $z \in C$ ,  $\text{past}(z, G) \subset C$  denotes the subset of blocks reachable from  $z$ , and similarly  $\text{future}(z, G) \subset C$  denotes the subset of blocks from which  $z$  is reachable (these are blocks that are provably created before after after  $z$  correspondingly). Note that an edge in the DAG points back in time, from the more recently created block to the one it extends. We denote by  $\text{anticone}(z, G)$  the set of blocks that the DAG does not directly ordered compared to  $z$ , that is:  $G \setminus (\text{past}(z, G) \cup \{z\} \cup \text{future}(z, G))$ . Additional notation includes:  $\overline{\text{past}}(z, G) := \text{past}(z, G) \cup \{z\}$ ,  $\text{antipast}(z, G) := C \setminus \overline{\text{past}}(z, G)$ , and similarly for the future and anticone sets. To avoid cumbersome notation we will frequently abuse notation and write  $z \in G$  instead of  $z \in C$ . The notation  $|G|$  represents the number of blocks in  $C$ . The unique block *genesis* is the block created at the inception of the system, and every valid block must have it in its past set.

An honest node  $v$  does not consider block  $b$  as valid (or as received by it) if it hasn't received first  $\text{past}(z, G)$ . Moreover, no one can modify the contents of a block or of its header once it has been created (as these will invalidate the PoW, and the block's hash). These two facts imply that the set  $\text{past}(b, G)$ , in sharp contrast to  $\text{future}(z, G)$  and  $\text{anticone}(z, G)$ , is fully determined by  $b$  and does not require specifying a “context”  $G$ . Accordingly, we will most of the time omit  $G$  and simply write  $\text{past}(b)$  (we may also write  $\text{future}(z)$ , if the context is of no importance).

Finally, for any DAG  $G$  we relate to a hypothetical block, which we denote  $\text{virtual}(G)$ . This block satisfies  $\text{past}(\text{virtual}(G)) = G$ . While its role is merely methodological, the virtual block of  $G$  can also be thought of as representing the next

---

<sup>10</sup>To avoid cumbersome notation, we assume a valid block does not contain two conflicting transactions. In practice, the order between such two transaction can be decided by the order in which they appear in the block.

block that an honest node whose current observed DAG is  $G$  attempts to create.

**Message broadcasting.** Nodes are instructed to send to their network peers all blocks that they have created as well as blocks they have received from other peers. We denote by  $publication(b)$  the time at which some node had *begun* the transmission of  $b$  to some other honest node. We denote by  $received^v(b)$  the time at which node  $v$  received  $b$ . We assume the existence of a constant  $D > 0$  such that any message (of size  $\leq B$ ) transmitted by an honest node arrives after at most  $D$  seconds at nodes that possess together at least  $(1 - \alpha)$  of the computational power, regardless of any manipulation of the attacker. Every time a user estimates the robustness of transactions in the ledger, SPECTRE’s *ChkRobustAccept* procedure requires he specifies an upper bound on the recent  $D$  in the network. In the analysis below we denote this estimate by  $d$  (see Footnote 12). Miners, however, do not require any knowledge of the value of  $D$ .

**The miner protocol.** Denote by  $G_t^v$  the DAG that node  $v \in \mathcal{N}$  observes at time  $t$ . The requirement from a given node  $v$  to broadcast all blocks upon receiving or creating them, is given formally as:  $\forall u \in \mathcal{N}: G_{t+D}^u \supseteq G_t^v$ . The requirement that  $v$  list in its new block all leaf-blocks of its current DAG is given formally as:  $\forall b$  with  $v = node(b): \overline{past}(b) = G_{time(b)}^v$ . To avoid redundancy, we additionally require that the list of pointers in  $b$ ’s header will include only leaf-blocks of  $past(b)$ . The set *honest* is defined formally by all nodes that at any time satisfy these requirements, and the set *attacker* consists of the rest of the nodes. We will occasionally abuse notation and use these sets to denote blocks created by these types of nodes (e.g.,  $b \in honest$  reads  $node(b) \in honest$ ). The DAG  $G_t^{pub}$  is defined by  $\cup_{u \in honest} G_t^u$ , and the DAG  $G_t^{oracle}$  is defined by  $\cup_{u \in \mathcal{N}} G_t^u$ .

**Proof-of-work.** In order to be considered valid, a block must contain a proof-of-work in the form of a valid nonce, as is used in Bitcoin. This nonce should have the property that it concatenates together with the block header into a string whose cryptographic hash is smaller than some predefined constant *TARGET*. This task is assumed to be solvable only by brute force, hence requires considerable computational power. Ordinary machines perform countless hashes per second, and the probability that the next guess of a nonce would be valid follows a memoryless distribution. Accordingly, the block creation process is well-approximated by a Poisson distribution. We denote by  $\lambda$  the rate of block creation in the system, when all nodes in the network participate.

In addition to  $d$ , *ChkRobustAccept* requires that the user specify  $\alpha$  – the fraction of computational power controlled by the attacker. Formally,  $d$  should be the time it takes messages to propagate to honest nodes that control at least a fraction  $1 - \alpha$  of the total hashrate.<sup>11</sup> The implication of the user’s *beliefs* regarding  $d$  and  $\alpha$  is that the guarantees provided by our algorithms hold only if the parameters it provides as inputs are indeed *upper bounds* on their actual (unknown) values.<sup>12</sup>

We stress, again, that *the miners operate irrespective of any assumption on the network’s propagation delay, its total hashrate, or the hashrate of dishonest nodes.*

<sup>11</sup>The remaining hash rate by more distant honest nodes is attributed to the attacker in this manner as a form of worst-case assumption.

<sup>12</sup>In fact, it is sufficient that the recent value of  $D \cdot \lambda$  is upper bounded by the user’s belief regarding this product. This parameter captures the average number of blocks created per unit of delay.



**Desired properties of a distributed ledger.** The *GetAccepted* procedure takes as input a block DAG  $G$  and returns a consistent subset of the transactions contained within it.

The *ChkRobustAccept* procedure, is used to determine if a transaction in this set is likely to remain there. *ChkRobustAccept* takes as input a block DAG  $G$ , a transaction  $tx \in \text{GetAccepted}(G)$ , and an  $\epsilon$ , and returns ACCEPT or WAIT. When  $G = G_t^v$  and *ChkRobustAccept* returned ACCEPT, we say that  $v$  has  $\epsilon$ -accepted  $tx$  at time  $t$ . The following properties are desirable:

**Property 1** (Consistency). The accepted set is consistent. For any DAG  $G$ ,

1. if  $tx \in \text{GetAccepted}(G)$  and  $tx_2 \in \text{inputs}(tx)$  then  $tx_2 \in \text{GetAccepted}(G)$ .
2. if  $tx \in \text{GetAccepted}(G)$  and  $tx_2 \in \text{conflict}(tx)$  then  $tx_2 \notin \text{GetAccepted}(G)$ .

**Property 2** (Safety). If some node  $\epsilon$ -accepts, then all other honest nodes will do so (w.p.  $1 - \epsilon$ ): For any  $v \in \text{honest}$ , if  $\text{ChkRobustAccept}(tx, \epsilon, G_t^v) = \text{ACCEPT}$  then, with probability of at least  $(1 - \epsilon)$ , there exists a  $\tau$  of finite expectation such that  $\forall u \in \text{honest}, \forall s \geq \tau : \text{ChkRobustAccept}(tx, \epsilon, G_s^u) = \text{ACCEPT}$ .

**Property 3** (Progress). Transactions become more robust as time passes: For any  $v \in \text{honest}$ , if  $\text{ChkRobustAccept}(tx, \epsilon, G_t^v) = \text{ACCEPT}$  then, with probability of  $(1 - \epsilon)$  at least, for any  $\epsilon'$  there exists a  $\phi$  of finite expectation such that  $\forall s \geq \phi : \text{ChkRobustAccept}(tx, \epsilon, G_s^u) = \text{ACCEPT}$ .

**Property 4** (Weak Liveness). Transactions are  $\epsilon$ -accepted in finite time (provided that their inputs are  $\epsilon$ -accepted, and there are no conflicts visible): Let  $t$  be the current time, and assume that  $tx \in x \in G_t^{\text{pub}}$ . Let  $\psi$  be the first time at which an honest node  $\epsilon$ -accepts  $tx$ . Then, conditioned on the event where  $\text{conflict}(tx) \cap G_\psi^{\text{pub}} = \emptyset$  and on the event where for all  $tx_2 \in \text{inputs}(tx)$ ,  $tx_2$  remains  $\epsilon$ -accepted forever (by some honest node), the expectation of  $\psi$  is finite.

## 5 The SPECTRE Protocol (formal)

We turn to describe the formal procedures of SPECTRE. In Section 2 we described informally the way SPECTRE pairwise orders blocks, and how this order is used to construct the subset of accepted transactions. We describe this construction formally in the first subsection below, thereby implementing the *GetAccepted* procedure. In the second subsection we implement the *ChkRobustAccept* procedure, which measures how robustly is a given transaction accepted.

### 5.1 Implementation of *GetAccepted*

**The accepted set of transactions.** We begin with a precise implementation of the pairwise ordering of blocks, described informally in Section 2.

In the algorithm,  $\widetilde{\text{sgn}}(n) = -1$  for  $n < 0$ ,  $\widetilde{\text{sgn}}(n) = +1$  for  $n > 0$ , and  $\widetilde{\text{sgn}}(0) = 0$ . To see that the recursion calls from line 4 halt observe that they take as inputs DAGs

---

**Algorithm 1** CalcVotes

---

**Input:**  $G$  – a block DAG

**Output:**  $vote(virtual(G))$  – a pairwise ordering of blocks in  $G$

```
1: if  $G = \emptyset$  then
2:   return an empty function
3: for all  $z \in G$  do
4:    $vote(z, past(z)) \leftarrow CalcVotes(past(z))$  and break ties arbitrarily
5: for all  $z \in G$  in some topological order (from leaves to root) do
6:   for all  $x, y \in G$  ( $x \neq y$ ) do
7:     if  $(x \in \overline{past}(z) \wedge y \in \overline{antipast}(z)) \vee (x \in past(z), y = z)$  then
8:        $vote_{x,y}(z, G) \leftarrow -1$ 
9:     else if  $(y \in \overline{past}(z) \wedge x \in \overline{antipast}(z)) \vee (y \in past(z), x = z)$  then
10:       $vote_{x,y}(z, G) \leftarrow +1$ 
11:    else if  $x, y \in past(z)$  then
12:       $vote_{x,y}(z, G) \leftarrow vote_{x,y}(z, past(z))$ 
13:    else if  $x, y \in \overline{antipast}(z)$  then
14:       $vote_{x,y}(z, G) \leftarrow \widetilde{sgn}\left(\sum_{z' \in future(z, G)} vote_{x,y}(z', G)\right)$ 
15:  $vote(virtual(G), G) \leftarrow \widetilde{sgn}\left(\sum_{z \in G} vote(z, G)\right)$ 
16: return  $vote(virtual(G), G)$ 
```

---

strictly smaller than  $G$  (because  $past(z) \subsetneq G$ ), and hence eventually all arrive at the base case  $G = \emptyset$  and return.

**The accepted set of transactions.** Algorithm 2 outputs a set of accepted transactions. It operates recursively, and should be initially called with  $GetAccepted(G, G)$  (we later denote this simply by  $GetAccepted(G)$ ). In the algorithm, the notation  $Z_G(tx)$  stands for all blocks in  $G$  that contain  $tx$ . Some complexity arises due to possible multiple copies of the same transactions, which requires dealing with equivalence classes of transactions (see Section 4).

## 5.2 Implementation of *ChkRobustAccept*

We now turn to analyze how robustly a given transaction can be considered accepted. Identifying the robustly-accepted-transaction set is done similarly to the construction of the accepted-transaction set, as outlined in Subsection 2.3: First, we analyze the robustness of the block ordering; then, we use this to verify that a given transaction is robustly accepted.

**Robustness of the block pairwise ordering.** Algorithm 3 outputs an upper bound on the probability that an attacker will be able to reverse the relation  $x \prec y$ . When the argument  $y$  is unspecified, the interpretation of the algorithm's output is  $x$ 's robustness against an unseen block (withheld by an attacker or yet to be created). In the algorithm,  $gap(b, G)$  denotes the size of the set  $\{z \in \overline{anticone}(b, G) : vote_{z,b}(virtual(G)) \geq 0\}$ . The notation  $\langle G, z, K \rangle$  will be explained in the paragraphs that follow.

**Explanations about the operation of Algorithm 3.** In line 13 the algorithms

---

**Algorithm 2** *GetAccepted*

---

**Input:**  $G$  – a block DAG,  $subG$  – a subDAG of  $G$  which is the past of a (possibly virtual) block

**Output:**  $TX$  – a hyper-set of valid transactions in  $G$

```
1:  $TX \leftarrow \emptyset$ 
2: for all  $z_1 \in G \cap subG$ 
   (in a leaves-to-root topological order) do
3:   for all  $tx \in z_1$  do
4:     for all  $tx_2 \in G \cap conflict(tx)$  do
5:       for all  $z_2 \in Z_G(tx_2) \cap anticone(z_1, G)$  do
6:         if  $vote_{z_1, z_2}(virtual(G)) \geq 0$  then
7:           break (to line 3 and pick next  $tx$ )
8:         if  $[tx_2] \cap GetAccepted(G, past(z_1)) \neq \emptyset$  then
9:           break (to line 3 and pick next  $tx$ )
10:        for all  $[tx_3] \in inputs(tx)$  do
11:          if  $[tx_3] \cap GetAccepted(G, past(z_1)) = \emptyset$  then
12:            break (to line 3 and pick next  $tx$ )
13:        add  $tx$  to  $TX$ 
14: return  $TX$ 
```

---

---

**Algorithm 3** *Risk* (offline)

---

**Input:**  $G$  – a block DAG,  $x$  – a block in  $G$ ,  $y$  (optional) – a block in  $anticone(x, G)$

**Output:**  $risk$  – an upper bound on the probability of block  $x$  not defeating  $y$  at any point in the future

```
1: if  $time\_now < publication(x) + 2 \cdot d$  then
2:   return 1
3:  $K \leftarrow \lceil \sqrt{|future(x, G)|} \rceil$ 
4: if  $NULL = y$  then
5:    $g \leftarrow |future(x, G)|$ 
6:    $M \leftarrow 0$ 
7: else
8:    $g \leftarrow \sum_{z' \in \overline{future}(x, G)} vote_{y, x}(z', G)$ 
9:    $M \leftarrow \left| \left\{ z \in future(x, G) : vote_{x, y}(z, G) = +1 \wedge gap(z, \langle G, z, K \rangle) = 0 \right\} \right|$ 
10:  $n_x \leftarrow |future(x, G)| - M$ 
11:  $j \leftarrow gap(x, G) + K$ 
12:  $l \leftarrow K$ 
13:  $risk \leftarrow f_{pre\_mine}(l) + f_{pre\_pub}(K) + f_{post\_pub}(M) + f_{post\_mine}(n_x, g, j, l, M)$ 
14: return  $risk$ 
```

---

uses some functions whose precise definitions we defer to later sections. An explicit formula for  $f_{pre\_pub}$  is given in (54), for  $f_{post\_pub}$  is given in (50) and (52), and for  $f_{post\_mine}$  is given in (2). Preceding Lemma 23, we provide a method to calculate

$f_{pre\_mine}$  numerically.

Intuitively, the function  $f_{pre\_mine}$  upper bounds the probability that the attacker has gained an advantage larger than  $l$  during the pre-mining phase (i.e., up until the creation of  $x$ ). The function  $f_{post\_mine}$  upper bounds the probability that the attacker will ever be able to create enough blocks so as to reverse the relation  $x \prec y$ . In essence,  $f_{post\_mine}$  is an adaptation of a formula from [17]. According to our version of the formula, if during the interval  $[time(x), t]$  (where  $t$  represents the current time) honest nodes created  $n$  blocks, then  $\binom{n-1+m}{m} \cdot \alpha^m \cdot (1-\alpha)^n$  is the probability that the attacker has created during this interval  $m$  blocks. If  $g$  aggregates all the votes of blocks in  $future(x, G)$ , then the probability that the attacker will be able to reverse the majority's vote is roughly  $\left(\frac{\alpha}{1-\alpha}\right)^{\max\{g-m, 0\}}$ . The combined expressions produce an upper bound on the success-probability of an attack.<sup>13</sup>

The main challenge here is to correctly measure  $n$ . This is a difficult task, as Algorithm 3 uses only structural information,<sup>14</sup> and does not rely on measurements of blocks' timings. Naïvely one would use  $n \approx |future(x, G)|$  to upper bound blocks created after  $publication(x)$ . However, there are two main difficulties:

- The block  $x$  might have been created by a dishonest node and withheld by it. In this case, there might have passed a long time between its creation and its publication, which implies that  $|future(x, G)|$  alone may be well below  $n$ . To avoid underestimating  $n$ , we upper bound the number of honest blocks in  $anticone(x, G)$ , by the variable  $j$ , and add it to our count (the addition is done inside  $f_{post\_mine}(n_x, g, j, l, M)$ ).

The function  $f_{pre\_pub}$  upper bounds the probability that we have underestimated  $j$ .

- By publishing his attack blocks, the attacker can increase the size of  $future(x, G)$  and cause us to overestimate  $n$ . This in turn result in an upper bound on the success-probability of an attack that is not tight enough, which would allow an attacker with a large value of  $\alpha$  to delay acceptance indefinitely. *Risk* overcomes this problem, by recognizing attacker blocks and excluding them from the count of  $n$ . This is done as follows.

Let  $G$  be a block DAG,  $b$  a block in  $G$ , and  $K$  a whole number. The DAG  $\langle G, b, K \rangle$  is obtained by creating a new chain  $z_1, \dots, z_K$  of  $K$  hypothetical blocks, connecting an edge from  $z_1$  to  $b$  and replacing every edge  $(z, b) \in G$  with  $(z, z_K) \in G$ . Essentially, this adds to the DAG  $K$  artificial voters which vote strongly  $x \prec y$ , against any  $y \in antipast(x, G)$ . In line 9, the algorithm checks whether  $gap(z, \langle G, z, K \rangle) = 0$ , i.e., whether there exists a block in  $anticone(z)$

<sup>13</sup>The calculations we use are quite more involved, as will be detailed later on. In particular, the reason why we aggregate in  $g$  votes from  $future(x, G)$  alone – rather than votes from the entire DAG – will become clearer in Sections 8 and 10. For now, observe that counting all votes – including weak votes – is not meaningful, as weak voters might reverse their vote as future events unfold. Rather, it is useful to measure how robustly weak voters support  $x \prec y$ , which is captured by our following procedure.

<sup>14</sup>With the exception of making sure that  $x$  has been published for at least  $2 \cdot d$  seconds.

that precedes  $z$  in the modified DAG  $\langle G, z, K \rangle$ . In the case of a negative answer,  $z$  is counted into  $n$  (in line 10).

Indeed, SPECTRE admits the following property: *If we add  $k$  voters in favour of an honest block, for some small  $k$ , then no other block will precede it in the pairwise ordering (apart from its past set).* This property is stated formally and proven in Lemma 28.

The function  $f_{post\_pub}$  upper bounds the probability that we have underestimated the number of honest blocks in  $future(x, G)$ .

We refer the reader to Section 8 for further understanding of the operation of *Risk*.

**Robustness of transaction acceptance.** The next step is to translate robustness of blocks (calculated by *Risk*) to robustness of transactions. This transition is implemented similarly to the transition from the (non-robust) ordering of blocks (Algorithm 1) to the (non-robust) accepted set of transactions (Algorithm 2).

The main procedure in this layer is *RiskTxAccept*. It takes as input  $G$  and  $tx$  (and an additional argument) and returns an upper bound on the probability that some honest node will not  $\epsilon$ -accept  $tx$ .

We now describe the main procedure of the second layer, *RiskTxAccept*, which utilizes and translates the robustness of blocks into robustness of transactions. The main task of *RiskTxAccept* is to properly account and aggregate the error bounds that *Risk* induces. As can be easily recognized, *RiskTxAccept* and *RiskTxReject* are mirror images of each other. While *RiskTxAccept* upper bounds the probability that a given transaction will ever be *removed* from the accepted transaction subset, *RiskTxReject* upper bounds the probability that a given transaction will ever be *included* in this subset. This is particularly vital for the case where two conflicting transactions are related topologically, i.e.,  $tx_2 \in y$  and  $tx_1 \in x \in future(y)$ , but  $tx_2$  is not in the accepted set (due to some previous conflict). In this case, although the block containing  $tx_2$  precedes that containing  $tx_1$ , we accept  $tx_1$ . It can be further  $\epsilon$ -accepted if the rejection-status of  $tx_2$  is robust, as calculated by *RiskTxReject*.

---

**Algorithm 4** *RiskTxAccept*

---

**Input:**  $G$  – a block DAG,  $subG$  – a subDAG of  $G$  which is the past of a (possibly virtual) block,  $tx$  – a copy of the transaction to defend

**Output:**  $risk$  – an upper bound on the probability that some honest node in some future point in time would not accept any no transaction in  $[tx] \cap subG$ .

```
1:  $minrisk \leftarrow 1$ 
2: for all  $z_1 \in Z_G([tx] \cap subG)$  do
3:    $risk \leftarrow Risk(G, z_1, \emptyset)$ 
4:   for all  $tx_2 \in G \cap conflict(tx)$  do
5:     for all  $z_2 \in Z_G(tx_2) \cap anticone(z_1, G)$  do
6:        $risk \leftarrow risk + Risk(G, z_1, z_2)$ 
7:        $risk \leftarrow risk + RiskTxReject(G, [tx_2], past(z_1))$ 
8:     for all  $[tx_3] \in inputs(tx) \cap past(z_1)$  do
9:        $risk \leftarrow risk + RiskTxAccept(G, [tx_3], past(z_1))$ 
10:   $minrisk \leftarrow \min\{minrisk, risk\}$ 
11:  $risk \leftarrow minrisk$ 
12: return  $risk$ 
```

---

---

**Algorithm 5** *RiskTxReject*

---

**Input:**  $G$  – a block DAG,  $subG$  – a subDAG of  $G$  which is the past of a (possibly virtual) block,  $tx$  – a copy of the transaction to defend

**Output:**  $risk$  – an upper bound on the probability that some honest node in some future point in time would accept a transaction in  $[tx] \cap subG$ .

```
1:  $risk \leftarrow 0$ 
2: for all  $z_1 \in Z_G([tx] \cap subG)$  do
3:    $minrisk \leftarrow 1$ 
4:   for all  $tx_2 \in G \cap conflict(tx)$  do
5:     for all  $z_2 \in Z_G(tx_2) \cap anticone(z_1, G)$  do
6:        $minrisk \leftarrow \min\{minrisk, Risk(G, z_2, z_1)\}$ 
7:        $minrisk \leftarrow \min\{minrisk, RiskTxAccept(G, [tx_2], past(z_1))\}$ 
8:     for all  $[tx_3] \in inputs(tx)$  do
9:        $minrisk \leftarrow \min\{minrisk, RiskTxReject(G, [tx_3], past(z_1))\}$ 
10:   $risk \leftarrow risk + minrisk$ 
11: return  $risk$ 
```

---

Building on these procedures, we now present the  $\epsilon$ -acceptance policy of SPECTRE. The user should provide as input the entire DAG that it currently observes.

---

**Algorithm 6** *ChkRobustAccept*

---

**Input:**  $G$  – a block DAG representing the current DAG observed by the node running the policy,  $tx$  – the transaction to defend,  $\epsilon$  – the maximum risk the user is willing to tolerate,  $\alpha$  – maximal size of attacker,  $d$  – maximal delay diameter of network,  $\lambda$  – the block creation rate

**Output:** ACCEPT– if the probability that (after some point in time) some honest node will not  $\epsilon$ -accept  $tx$  is smaller than  $\epsilon$ ; WAIT– otherwise

```
1: if  $RiskTxAccept(G, [tx] \cap G) < \epsilon$  then  
2:   return ACCEPT  
3: else  
4:   return WAIT
```

---

### 5.3 Formal claim

Having described SPECTRE’s structure and main procedures we are ready to restate the main theorem given in Section 1:

**Theorem 1.** *For any  $\alpha < 0.5$  and for any  $d \cdot \lambda$ , SPECTRE’s Algorithm 2 and 6 satisfy Properties 1-4.*

Moreover, the expected values of  $\tau, \phi, \psi$  are in  $\mathcal{O}\left(\frac{\ln(1/\epsilon)}{\lambda(1-2\alpha)} + \frac{d}{1-2\alpha}\right)$ . The proof can be found in Section 10.

### 5.4 Online policy

We now present an alternative implementation of *Risk*, which requires that the user be online at the time when his block gains confirmations. This assumption is highly reasonable for many practical scenarios, e.g., a cashier serving a continuous line of customers. The main benefit of the online version is that it relies on a tighter analysis, and therefore accepts transactions slightly faster. We now confine ourselves to the case where there is no visible double-spend (i.e.,  $y = NULL$ ).

The fact that the user is online can be utilized in two ways: First, any block that the user receives after  $received^v(b) + 2 \cdot d$  and does not belong to  $future(x)$  can be marked by him as an attacker block. Second, the user can estimate the number of hidden attacker blocks by measuring the time that passed since the creation of  $x$ .

Below we describe the online version of *Risk*. The algorithm takes as input node  $v$ ’s DAG and the block  $x$  to defend, and returns an upper bound on the probability that some block  $y \in G_\infty^{pub} \setminus G_t^{pub}$  will ever precede it.

---

**Algorithm 7** *Risk* (online)

---

**Input:**  $G_t^v$  – the block DAG that  $v$  observes at time  $t$ ,  $x$  – a block in  $G_t^v$   
**Output:**  $risk$  – an upper bound on the probability of block  $x$  not defeating  $y$  at any point in the future, for some  $y \in G_\infty^{pub} \setminus G_t^{pub}$

- 1: **if**  $time\_now < publication(x) + d$  **then**
- 2:     **return** 1
- 3:  $T \leftarrow time\_now - received^v(x)$
- 4:  $G_x \leftarrow G_{received^v(x)+2 \cdot d}^v \cup future(x, G_x)$
- 5:  $g \leftarrow \min_{x' \in \overline{anticone}(x, G_x)} |future(x', G_x)|$
- 6:  $risk \leftarrow risk\_hidden(T, g)$
- 7: **if**  $risk < \epsilon$  **then**
- 8:     **return** *ACCEPT*
- 9: **else**
- 10:    **return** *WAIT*

---

The definition of *risk\_hidden* appears in (45)-(46). In practice, as node  $v$  may have a partial view of  $G_\infty^{pub} \setminus G_t^{pub}$ , in order to use Algorithm 7 the user must wait additional  $d$  seconds and verify that  $conflict(tx) \cap G_{t+d}^v = \emptyset$ , i.e., that the attacker did not publish a double-spend in the interval  $[t-d, t]$ . The correctness of the online policy modification is proven in Corollary 26.

## 6 Simulation results

We implemented the SPECTRE protocol in Python along with an event-driven simulator of network dynamics. For each experiment we generated an Erdős-Rényi random network topology with 20 nodes. Each node forms 5 outgoing links, in expectation. The delay on each link was uniformly distributed and later scaled linearly so that the diameter of the graph is  $d$  (for the given  $d$ ). Every point represents the average outcome over at least 500 experiments.

The main benefit of SPECTRE is fast transaction confirmation. In order to measure the waiting times in SPECTRE, we utilized the online acceptance policy derived by Algorithm 7. Accordingly, we stress that the merchant needs to wait additional  $d$  seconds in order to verify that no double-spend has been released in the past  $d$  seconds, as explained at the end of Section 5.

**How does the delay diameter affect acceptance times?** Given that block creation rate is high, most of the waiting time for acceptance is dominated by the block propagation delay. Fig. 5 depicts the transaction acceptance times of SPECTRE, for various values of the delay diameter  $d$ , and for different security thresholds  $\epsilon$ . Note that, unlike the Nakamoto consensus,  $d$  affects the acceptance time of transactions but not their security.



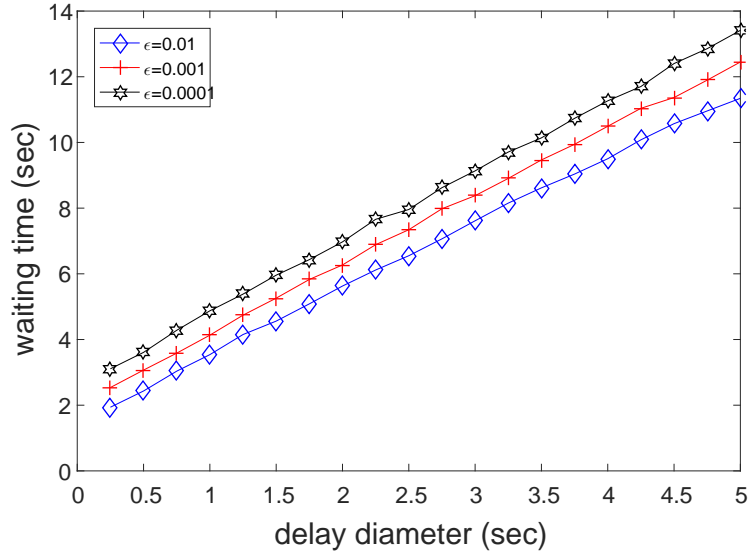


Figure 5: The average time for *ChkRobustAccept* to return ACCEPT, assuming there's no visible double-spending, for  $\lambda = 10$  blocks per second and  $\alpha = 0.25$ .

**How does the block creation rate affect acceptance times?** Fig. 6 depicts the acceptance times for various values of the block creation rate  $\lambda$ , under a constant delay  $d = 5$  seconds. The graph reaffirms the role of  $\lambda$  in our asymptotic bound: accelerating the block creation process allows for faster acceptance times. For comparison, Bitcoin's block creation rate of  $1/600$  implies waiting times that are orders of magnitudes higher (not plotted).

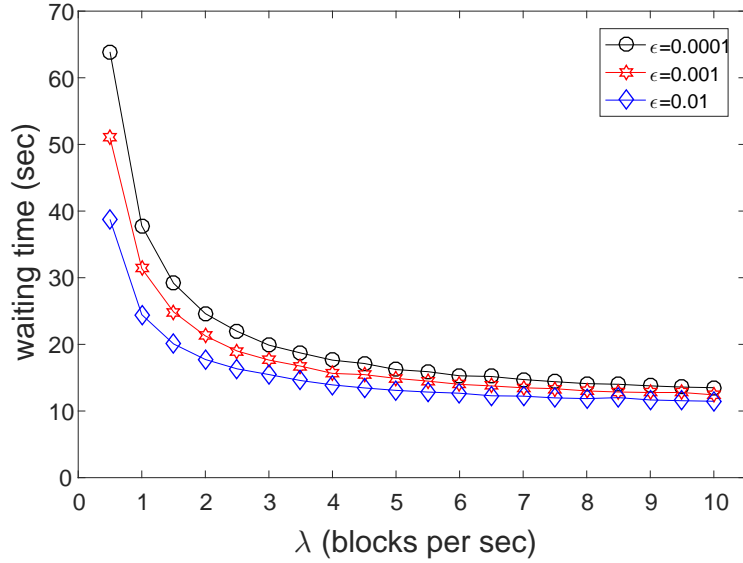


Figure 6: The average time for *ChkRobustAccept* to return ACCEPT, assuming there’s no visible double-spending, for  $d = 5$  seconds and  $\alpha = 0.25$ .

**Can an attacker delay acceptance?** We now turn to demonstrate the effect of censorship attacks in which some dishonest nodes publish blocks that do not reference other miners’ blocks. Recall that the Weak Liveness property of SPECTRE (Proposition 5) guarantees fast acceptance of transactions that are not visibly double-spent—even in the presence of a censorship attack. However, such an attack still causes some delay in transaction acceptance, but this delay is minor for small attackers. In Fig. 7 we quantify this effect, by comparing the acceptance times in “peace days” to those under an active censorship attack. The parameters here are  $d = 5$  seconds,  $\lambda = 10$  blocks per second, and  $\epsilon = 0.01$ . The results display a modest effect of the attack, and they show that in order to delay transaction acceptance by more than 5 to 10 seconds an attacker must possess a significant share of the computational power in the network.

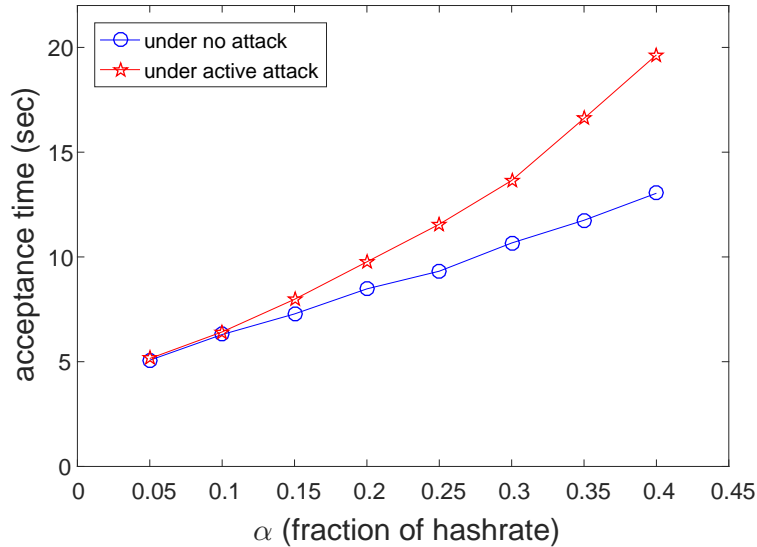


Figure 7: The average time for *ChkRobustAccept* to return ACCEPT, assuming there’s no visible double-spending, for  $d = 5$  seconds,  $\lambda = 10$  blocks per second, and  $\epsilon = 0.01$ , in the presence and in the absence of a censorship attack.

**How does  $\epsilon$  decrease for various sizes of the attacker?** Once an honest node  $\epsilon$ -accepts a transaction, there’s still a small risk ( $\epsilon$ ) that it would eventually be rejected. We show that the probability of this event vanishes quickly, even for an extremely capable attacker (e.g., with  $\alpha = 0.4$  of the hashrate). This is illustrated in Fig. 8, assuming  $d = 5$  seconds and  $\lambda = 10$  blocks per second (notice that the y-axis is in log scale).

**How tight is our security analysis?** The analysis on which Algorithm 3 relies makes several worst-case assumptions in order to bound the probability of a successful attack, e.g., that the attacker can broadcast blocks to and receive blocks from all nodes without any delay (see Section 10, mainly Lemmas 12 and 19). Accordingly, the analysis is not tight, and in reality attacks are in fact less likely to succeed. In Fig. 9, we depict the comparison between the analytical bound and two different empirical simulations. In these simulations we explicitly generate blocks for the attacker and simulate the optimal double-spending attack. We repeat the experiment 10,000 times for each point in the graph, and measure the empirical success rate. The simulations assume two types of attackers: a worst-case attacker that is able to transmit and receive blocks with no delays, and a more realistic attacker that is connected to other nodes with typical delays. We compared the fraction of successful attacks under these setups to the analytical risk calculated by SPECTRE’s policy (Algorithm 7).

The results show that the risk considered by SPECTRE’s *ChkRobustAccept* indeed upper bounds the actual risk, and that transactions are even safer than we guarantee formally.

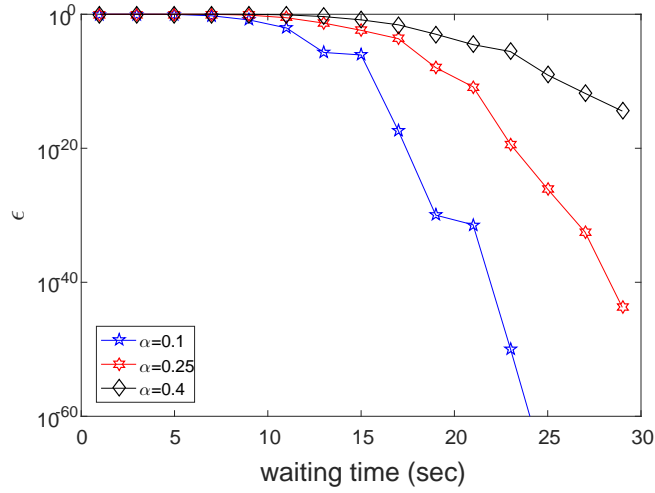


Figure 8: The probability of a successful double-spending attack, as a function of the waiting time before acceptance, under  $d = 5$  seconds and  $\lambda = 10$  blocks per second, for  $\alpha = 0.1, 0.25,$  and  $0.4$ . The probability here is the result of the calculation performed by Algorithm 3.

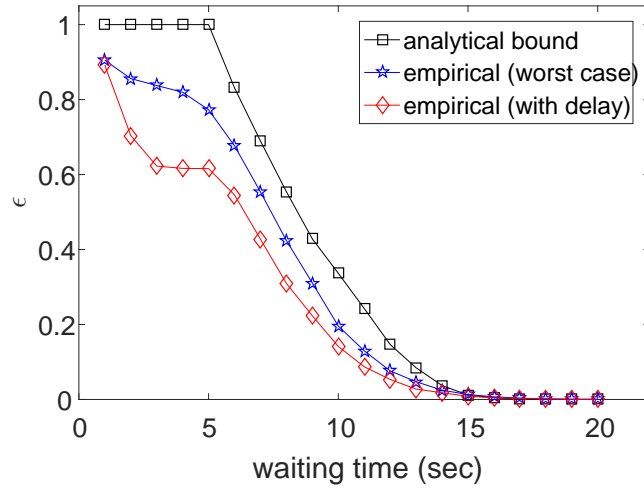


Figure 9: The analytical vs. empirical probabilities of a successful double-spending attack, as a function of the waiting time before acceptance, under  $d = 5$  seconds,  $\lambda = 10$ , and  $\alpha = 0.25$ .

## 7 Implementation Details

**Minting.** In SPECTRE, any block whose target meets the required value  $TARGET$  – as will be defined below – receives the same minting reward. If its target is higher

than  $TARGET$  (i.e., it is solved with an easier difficulty) by a factor of  $(1 + \delta)$  at most, then its reward is reduced by the same factor. The parameter  $\delta$  represents the protocol’s tolerance to blocks mined with an outdated difficulty. Thus, if for instance  $\delta$  is chosen to equal 2, then blocks with a target value of  $2 \cdot TARGET$  or  $3 \cdot TARGET$  are valid, and their minting rewards are reduced by a factor of 2 or 3, respectively; blocks with a target higher than  $3 \cdot TARGET$  are invalid and discarded. We now explain how  $TARGET$  is defined and readjusted.

**Retargeting.** Similarly to Bitcoin and other PoW-based systems, the difficulty of block creation, represented by  $TARGET$  (Section 4), must be occasionally adapted. Varying network conditions, and changes in the amount of computational resources invested in the system, require we limit the number of blocks created per second, to avoid network congestion. In Bitcoin this is done as follows: Every 2016 blocks, the next block – which we call the *reference block* – is mined according to an adjusted difficulty. The new difficulty is obtained by taking the time that elapsed since the previous reference block (using the timestamps written inside each block) and plugging it into the retargeting formula. The output of this formula is the new value of  $TARGET$  that the new reference block should be mined with.

We adapt this scheme to SPECTRE: Let  $x_{n-1}$  be the previous reference block. Every new block  $x_n$  that has the property that  $|past(x_n) \cap future(x_{n-1})| = 2016$ , is a candidate to become the new reference block. In case additional candidates exist, we choose the one with the minimal  $dist\_gap$  with some arbitrary tie-breaking (recall that  $dist\_gap(b, G)$  is the minimal  $k$  such that adding  $k$  votes in favour of  $b$  makes its  $gap$  equal zero, i.e.,  $gap(b, \langle G, b, k \rangle) = 0$ ). Thus, for  $x_n$  satisfying the above property, one and only block from  $\overline{anticone}(x_n)$  would be chosen as the reference block succeeding  $x_{n-1}$ . In particular, as explained in Section 3, an attacker block that was withheld for a while will have a large  $dist\_gap$  and will not be eligible as a reference block. Furthermore, an attacker block that was mined before  $x_{n-1}$  will not affect the next retargeting, as it cannot belong to  $future(x_{n-1})$ .

The new difficulty, with which the new reference block should be mined, is given again through the formula that uses the time that elapsed between  $x_{n-1}$  and  $x_n$  to update  $TARGET$ . **The formula should aim for a predefined  $\lambda$  for which nodes are believed to have sufficient bandwidth**, e.g., 1 MB per second. This difficulty dictates the difficulty for every block in  $antipast(x_n) \setminus \overline{anticone}(x_{n+1})$ , where  $x_{n+1}$  is the next reference block. Every block in this set should be mined according to the same difficulty as  $x_n$ .

If block  $b \in antipast(x_n) \setminus \overline{anticone}(x_{n+1})$  was solved with an easier difficulty than that dictated by the reference block  $x_n$ , then  $b$  is still considered valid, provided that its outdated target is at most  $(1 + \delta)$  of the target of  $x_n$  (i.e., a difficulty easier by at most  $(1 + \delta)$ ). The parameter  $\delta$  is the protocol’s tolerance threshold. The minting reward of  $b$  is reduced by the corresponding factor, as explained above. Blocks whose target exceeds the required one by a factor higher of  $(1 + \delta)$  are ignored and discarded.

**Block headers.** In order to incorporate all blocks into the DAG, every block embeds in its header pointers to the hash of previous blocks. No redundancies are permitted, hence only leaf-blocks of  $past(b)$  should be pointed at by the header of  $b$ . The implication of this is that a block’s header is of size  $\approx 50 + d \cdot \lambda \cdot 32$  Byte. Therefore,

there is a limit to the extent at which block size could be reduced and block creation rates increased – at extremely high rates, the overhead of the block header becomes significant relative to the number of included transactions. We note, additionally, that in case the current observable DAG has too many leaves (whether by a rare burst in block creations or by an attacker releasing many outdated blocks), the next block creator can cap the number of leaf-blocks it points at. Blocks left out by this block will later integrate into the DAG, as future blocks will have available space in their headers and will be able to point at these blocks and include them.

**Efficient implementation.** Our current implementation of SPECTRE uses naïve calculations which are usually inefficient, specifically, cascading the votes all the way to the *genesis* block. Several efficient implementations are possible. Designing such an efficient implementation requires attention to CPU attacks, in which the attacker exposes peculiar structures of outdated blocks in order to cause other nodes to perform extensive computation. It can be shown that these attacks are highly costly to the attacker. We leave the specification of an efficient implementation, and a proof of the cost of CPU attacks, to future work.

**Transaction fees.** The body of a transaction specifies the amount transferred from the payer to the payee. The transaction-fee specifies the payment from the payer to the miner whose block contains the transaction. We regard these two parts as separate transactions, in the following sense. Assume that  $tx \in x$ , and denote by  $fee(tx, x)$  the transaction representing the fee-payment of  $tx$  to the creator of block  $x$ . Assume now that two copies of  $tx$  appear in two different blocks  $x, y$ . Then the body is considered simply as a copy of the same transaction (recall the notation  $[tx]$  from Section 4), whereas the transactions  $fee(tx, x)$  and  $fee(tx, y)$  are considered a conflict, i.e., a double-spend. Accordingly, as in the ordinary scheme of SPECTRE, the fee is granted to (the creator of) block  $x$  iff  $tx \in GetAccepted(G)$  and  $x$  defeats all other blocks that contain  $tx$  as well.

This rule can potentially harm miners, in the special case when the relation between  $x$  and  $y$  does not become robust (SPECTRE does not guarantee robustness if these blocks were published in time proximity). We address this problem by introducing *settlement transactions*. A settlement transaction is a voluntary transaction which both the creators of  $x$  and  $y$  sign after they observe that their blocks conflict. We denote it  $settlement(x, y)$ . The interpretation of  $settlement(x, y)$  is that the fees from all of (or part of, if the parties involved so choose) the transactions in  $x \cap y$  should be divided evenly between blocks  $x$  and  $y$ .  $settlement(x, y)$  essentially overrides  $fee(tx, x)$  and  $fee(tx, y)$ . When  $settlement(x, y)$  appears in some block  $z$  in the DAG  $G$ , it is considered accepted (i.e., a member of  $GetAccepted(G)$ ) iff  $x, y \in past(z)$  and  $z$  precedes every block that contains a transaction *spending*  $fee(tx, x)$  or  $fee(tx, y)$ . Therefore, once one party has spent its fee before it belonged to it robustly, it won't be able to settle later (w.h.p.). Miners are therefore advised to wait for their transaction-fee rewards to become robust, or to initiate a settlement, before spending these rewards.

Note that this scheme can be used to settle conflicts between blocks of multiple parties simultaneously. Furthermore, the settlement scheme need not be confined to conflicts regarding fees, and can be applied to any double-spending.

## 8 Outline of the Proof

The proof of SPECTRE's correctness is involved and occasionally technical. In this section we outline the structure of the proof and state the propositions and lemmas we use. These are proved formally in Section 10.

### 8.1 Additional notation

- $\widehat{\mathcal{E}}_s^u(x, y) :=$  the event where  $vote_{y,x}(virtual(G_s^u)) = +1$ .
- $\mathcal{E}_s^u(x, y, \epsilon) :=$  the event where  $Risk(G_s^u, x, y) < \epsilon$ .
- $\widehat{\mathcal{A}}_s^u(tx) :=$  the event where  $tx \in GetAccepted(G_s^u)$ .
- $\mathcal{A}_t^u(tx, \epsilon) :=$  the event where  $ChkRobustAccept(G_t^u, tx, \epsilon) = ACCEPT$ .
- $\mathcal{E}_{t \rightarrow \infty}^{all}(x, y, \epsilon) :=$  the event  $\bigcap_{u \in honest} \bigcap_{s \in (t, \infty)} \mathcal{E}_s^u(x, y, \epsilon)$ , and similarly for  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$ ,  $\mathcal{A}_{t \rightarrow \infty}^{all}(tx, \epsilon)$ , and  $\widehat{\mathcal{A}}_{t \rightarrow \infty}^{all}(tx)$ .
- $past_h(z, G) := past(z, G) \cap honest$ , and similarly for the future and anticone sets.
- $V_{x \prec y}(G) := \{z \in G \mid z \text{ is a strong voter w.r.t. } (x, y) \text{ and } vote_{x,y}(z) = -1\}$ .
- $\mathcal{P}_{oiss}(\delta, j) := e^{-\delta} \cdot \frac{\delta^j}{j!}$ .

### 8.2 Formal claims

We now take apart Theorem 1 and write a separate proposition for each of the security properties Safety, Progress, and Weak Liveness, and for Consistency. By definition, proving Theorem 1 entails proving the following propositions:

**Proposition 2** (Consistency). *For any DAG  $G$ ,*

1. *if  $tx \in GetAccepted(G)$  and  $tx_2 \in inputs(tx)$  then  $tx_2 \in GetAccepted(G)$ .*
2. *if  $tx \in GetAccepted(G)$  and  $tx_2 \in conflict(tx)$  then  $tx_2 \notin GetAccepted(G)$ .*

**Proposition 3** (Safety). *For any  $v \in honest$ , if  $ChkRobustAccept(tx, \epsilon, G_t^v) = ACCEPT$  then, with probability of  $(1 - \epsilon)$  at least, there exists a  $\tau$  of finite expectation such that  $\forall u \in honest, \forall s \geq \tau : ChkRobustAccept(tx, \epsilon, G_s^u) = ACCEPT$ .*

**Proposition 4** (Progress). *For any  $v \in honest$ , if  $ChkRobustAccept(tx, \epsilon, G_t^v) = ACCEPT$  then, with probability of  $(1 - \epsilon)$  at least, for any  $\epsilon' < \epsilon$  there exists a  $\phi$  of finite expectation such that  $\forall s \geq \phi : ChkRobustAccept(tx, \epsilon, G_s^u) = ACCEPT$ .*

**Proposition 5** (Weak Liveness). *Let  $t$  be the current time, and assume that  $tx \in x \in G_t^{pub}$ . Let  $\psi$  be the first time at which an honest node  $\epsilon$ -accepts  $tx$ . Then, conditioned on the event where  $conflict(tx) \cap G_\psi^{pub} = \emptyset$  and on the event where for all  $tx_2 \in inputs(tx)$ ,  $tx_2$  remains  $\epsilon$ -accepted forever (by some honest node), the expectation of  $\psi$  is finite.*

These proposition assume an *arbitrary behaviour* of the attacker. To each of the last three propositions we write a matching one which regards robustness of blocks (rather than that of transactions).

**Proposition 6** (Safety (blocks)). *For any  $v \in \text{honest}$ , if  $\text{Risk}(x, y, G_t^v) < \epsilon$  then, with probability of  $(1 - \epsilon)$  at least, there exists a  $\tau$  of finite expectation such that  $\forall u \in \text{honest}, \forall s \geq \tau : \text{Risk}(x, y, G_s^u) < \epsilon$ .*

**Proposition 7** (Progress (blocks)). *For any  $v \in \text{honest}$ , if  $\text{Risk}(x, y, G_t^v) < \epsilon$  then, with probability of  $(1 - \epsilon)$  at least, for any  $\epsilon' < \epsilon$  there exists a  $\phi$  of finite expectation such that  $\forall s \geq \phi : \text{Risk}(x, y, G_s^u) < \epsilon$ .*

**Proposition 8** (Weak Liveness (blocks)). *Let  $t$  be the current time, and assume that  $x \in G_t^{\text{pub}}$ . Let  $\psi$  be the first time  $s$  at which for some honest node  $v$ :  $\text{Risk}(x, y, G_s^v) < \epsilon$ . Then, conditioned on the event where  $y \notin G_\psi^{\text{pub}}$ , the expectation of  $\psi$  is finite.*

### 8.3 Basic properties

The two following lemmas are immediate from lines 7-14 of Algorithm 1.

**Lemma 9.** *Topological relations are unanimously agreed: If  $G = (C, E)$  is a block DAG, and  $(y, x) \in E$ , then  $\forall z \in G : \text{vote}_{x,y}(z, G) = -1$ .*

**Lemma 10.** *A block's vote regarding block(s) in its past depends only on its past, hence remains fixed forever: Let  $G_1$  and  $G_2$  be two block DAGs, and assume  $x, y, z \in G_1 \cap G_2$ . If  $\{x, y\} \cap \overline{\text{past}}(z) \neq \emptyset$  then  $\text{vote}_{x,y}(z, G_1) = \text{vote}_{x,y}(z, G_2)$ .*

Accordingly, as mentioned in Section 2, we say that  $z$  is a strong voter w.r.t the pair  $(x, y)$  if  $z \in \overline{\text{future}}(x) \cup \overline{\text{future}}(y)$ , and otherwise it is a weak voter.

The following Lemma shows that the vote of the *genesis* coincides with the vote of the virtual block. Intuitively, the *genesis* votes according to the majority vote in the DAG excluding itself, and amplifies this majority, which in turn dictates the virtual block's vote.

**Lemma 11.** *genesis's vote is the final vote:  $\text{vote}(\text{virtual}(G)) = \text{vote}(\text{genesis}, G)$ .*

### 8.4 Proof of Safety of blocks (Proposition 6)

We turn to the proof of Proposition 6, namely, the claim that all nodes will forever agree on the (robustness of) the order  $x \prec y$ , provided that it was sufficiently robust in the DAG observed by some honest node. This is the main (and most involved) part of the proof. The rest of the propositions follow from it, and their proofs are rather self explanatory.

In order to simplify the analysis, we need to make some worst case assumptions regarding the behaviour of the attacker. Lemma 19 proves that these are indeed worst case assumptions, namely, that they indeed represent the optimal attack. The vote of each block under our modification is denoted  $p\text{-vote}()$ , a notion which we describe formally in Subsection 10.3.



In the next central lemma we show that, provided that the aggregate vote in  $future(x)$  is sufficiently biased in favour of  $x \prec y$ , the *genesis* block – hence the virtual block (by Lemma 11) – will vote  $x \prec y$ . This proves that, roughly speaking, *the vote of recent weak voters cascades through the DAG and convinces older weak blocks, forming thus the genesis's vote*. The way we prove this is by choosing a specific weak voter  $z_{late}$  (in case  $x$  is an honest block,  $z_{late} = x$ ), and making sure that its vote is sufficiently robust so as to guarantee that (i) it will not be reversed, and (ii) it will cascade all the way to the *genesis*. Consequently, a successful attack (namely, a reversal of  $x \prec y$  in the DAG observed by some honest node) requires that the attacker add more blocks to  $future(z_{late})$  than the honest network adds (up to some additive term), in some time interval.

The following lemma formalizes these observations. It uses some parameters ( $h$ ,  $j$ , etc.) that only an oracle can have full knowledge of. We will later show how in reality a node can infer the robustness of block relations without having access to these parameters.

**Lemma 12.** *Let  $t \geq publication(x) + 2 \cdot d$ . Let  $z_{late}$  be the latest block in  $\overline{past}_h(x)$ . Denote:*

- $h := |anticone_h(z_{late}, G_t^{oracle})|$
- $j := |future_h(z_{late}, G_t^{oracle}) \setminus future_h(x, G_t^v)|$
- $m := |future_a(z_{late}, G_t^{oracle}) \setminus future_a(x, G_t^v)|$
- $k_1 := |G_{[t-d, t]}^{oracle} \cap honest|$
- $l := \max_{z \in G_t^{oracle} \cap honest} \left\{ \left| future_a(z, G_{time(z_{late})}^u) \right| - \left| future_h(z, G_{time(z_{late})}^u) \right| \right\}$
- $g := \sum_{z \in \overline{future}(x, G_t^v)} vote_{y,x}(z, G_t^v)$

Then,

$$\begin{aligned} \widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^c &\subseteq \left\{ \exists s \geq t, \exists u \in honest \text{ s.t. } \left| G_{[t, s]}^u \cap attacker \right| \right. \\ &\geq \left. \left| G_{[t, s]}^u \cap honest \right| + g - 2 \cdot h - j - k_1 - l - m \right\}. \end{aligned}$$

Given the result of the previous lemma, we can upper bound the probability that the order of  $x \prec y$  will be reversed. This result resembles the conventional analysis of Bitcoin's security: The greater number of blocks currently pointing at  $x$  (and in SPECTRE: voting for  $x \prec y$ ), the less likely it is that the attacker will be able to win the block-count race and reverse the decision.

**Lemma 13.** *Given the parameters of Lemma 12,*

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathfrak{G}}\right) \leq \sum_{h'=0}^{\infty} \mathcal{P}_{oiss}(d \cdot (1 - \alpha) \cdot \lambda, h') \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{(g-2 \cdot h-j-k_1-l-m-h')^+}.$$

An ordinary node does not typically know for sure the values of the parameters assumed in Lemma 12. The next corollary shows that the result of that lemma (and the one that follows) applies when replacing these parameters with proper bounds thereof. We will later discuss how a node can obtain such bounds.

**Corollary 14.** *If*

- $j \geq |\text{anticone}_h(x, G_t^{\text{oracle}})|$
- $l \geq \max_{z \in G_t^{\text{oracle}} \cap \text{honest}} \left\{ \left| \text{future}_a(z, G_{\text{time}(z_{\text{late}})}^u) \right| - \left| \text{future}_h(z, G_{\text{time}(z_{\text{late}})}^u) \right| \right\}$
- $n_x \geq \text{future}_h(x, G_t^{\text{oracle}})$
- $g \leq \sum_{z \in \overline{\text{future}}(x, G_t^v)} \text{vote}_{y,x}(z, G_t^v)$ .

*Then*

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)\right) \leq \sum_{k=0}^{\infty} \mathcal{P}_{oiss}((2 - \alpha) \cdot d \cdot \lambda, k) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot (1 - \alpha) \cdot \lambda, h) \cdot \sum_{m=0}^{\infty} \binom{n_x + j + h + m - 1}{m} \cdot (1 - \alpha)^{n_x + j + h} \cdot \alpha^m \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{(g-2 \cdot h-k-j-l-m)^+} \quad (1)$$

We adjust the above results to the case where some blocks in  $\text{future}(x, G)$  are known to belong to the attacker. Here we assume that this knowledge is granted to us by a hypothetical oracle. Later on, we will see how attacker blocks are recognized by Algorithm 3, w.h.p.

**Corollary 15.** *If in addition to the assumptions of Corollary 14 we assume that*

$M \leq |future_a(x, G_t^v)|$ , then

$$\begin{aligned}
& \Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y) \mid |future_a(x, G_t^v)| \geq M\right) \leq \\
& \sum_{k=0}^{\infty} \mathcal{P}_{oiss}((2-\alpha) \cdot d \cdot \lambda, k) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot (1-\alpha) \cdot \lambda, h) \cdot \\
& \left( \sum_{m'=M}^{\infty} \binom{n_x + j + h + m' - 1}{m'} \cdot (1-\alpha)^{n_x + j + h} \cdot \alpha^{m'} \right)^{-1} \cdot \\
& \sum_{m=M}^{\infty} \binom{n_x + j + h + m - 1}{m} \cdot (1-\alpha)^{n_x + j + h} \cdot \alpha^m \cdot \\
& \left( \frac{\alpha}{1-\alpha} \right)^{(g-2 \cdot h - k - j - l - (m-M))^+}.
\end{aligned} \tag{2}$$

We denote the RHS of this inequality by  $f_{post\_mine}(n_x, g, j, l, M)$ .

So far, our analysis assumed that we are given some proper bounds over the parameters from Lemma 12. Lemmas 23, 28, and 30 show how to appropriately bound these parameters. For each of these parameters, a separate error function is defined, which upper bounds the probability that it does not serve as a correct bound. These error functions deteriorate exponentially fast, by Lemmas 24, 29, and 31. Algorithm 3 aggregates these error functions into the total risk that it outputs.

The parameters are:

- $l$  – the pre-mining lead that the attacker obtained before the publication of  $x$ , with error function  $f_{pre\_mine}(l(G_t^v))$ , calculated numerically in Subsection 10.3.1
- $n_x$  – the number of honest blocks in  $future(x, G_t^v)$ , with error function  $f_{post\_pub}(|future(x, G_t^v)|)$ , defined in Corollary 28 (Inequality (52)), and
- $j$  – the number of honest blocks created after  $time(x)$ , with error function  $f_{pre\_pub}(n_j(G_t^v))$ , defined in Lemma 30 (Inequality (54)).

While we have previously shown that  $n_x$  properly counts all honest blocks, we now show that it does successfully exclude almost all attack blocks. Without such a guarantee, weaker attackers would have been able to publish their blocks and delay acceptance indefinitely.

**Lemma 16.** *Conditioned on the event  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$ , there exists a time  $\tau \in [t, \infty)$  such that  $\forall s \geq \tau: M(oracle^u, s) \geq \left| future_a(x, G_s^{oracle^u}) \cap G_{[t,s]}^{oracle^u} \setminus V_{x \prec y}(G_s^{oracle^u}) \right| - m^*$ , for some  $m^*$  that remains fixed after  $\tau$  (and with  $\mathbb{E}[m^*]$  determined by the events up to time  $t$ ).*

The above analysis (particularly Lemma 12) has upper bounded the probability that the attacker would be able to reverse the relation  $x \prec y$ . We now show that, conditioned on the order remaining  $x \prec y$ , the error function  $f_{post\_mine}$  (which upper bounds the probability of this order ever reversing) vanishes as well, which in turn implies that their order would be considered robust by all honest nodes.

**Lemma 17.** *There exists a  $\psi \in [t, \infty)$  such that  $\Pr \left( \mathcal{E}_{t \rightarrow \infty}^{all}(x, y, \epsilon)^{\mathbb{G}} \mid \mathcal{E}_t^v(x, y, \epsilon) \right) < \epsilon$ . Moreover,  $\mathbb{E}[\psi] < \epsilon$ .*

We have thus shown that if the output of Algorithm 3, as run by some honest node, was smaller than  $\epsilon$  then with probability of at least  $1 - \epsilon$ , any honest node running Algorithm 3 (after some time) will get a result smaller than  $\epsilon$ .<sup>15</sup> This completes the proof of Safety w.r.t. blocks.

The same technique used in the proof of Lemma 17 is used to prove the Progress property (Proposition 7); indeed, in the proof we see that the term that aggregates all the error functions vanishes as time develops, w.h.p., in the perspective of all honest nodes. In particular, for  $v$  (the node that originally  $\epsilon$ -accepted the transaction), it becomes smaller than  $\epsilon'$  w.h.p. A similar usage of this argument is used to prove Weak Liveness (Proposition 8); indeed, in the latter we only need to regard the case where  $y = NULL$ . In this case, all published blocks are strong voters in favour of  $x$ , and so we can guarantee the convergence of the error functions without going through Lemma 12 and the analysis that follows.

The proofs of the Safety, Weak Liveness, and Progress w.r.t. transactions rely directly on their block-variants. We refer the reader to Section 10 to fill in these gaps.

## 9 Conclusions

In this work we presented SPECTRE, a new protocol for fast and scalable distributed ledgers. We have shown SPECTRE's ability to process a high rate of block creation, as well as to handle large blocks securely. Our results demonstrate that SPECTRE can achieve incredibly low confirmation times, especially compared to the Nakamoto consensus. Further work to improve and tighten the acceptance policy we derived can lower confirmation times further.

SPECTRE thus shows the feasibility of creating open, permissionless distributed ledgers that will work successfully at large scale. SPECTRE achieves these properties due to the fact that it is willing to delay double-spent transactions (possibly forever). This fact also makes SPECTRE less suitable for systems like Ethereum, where a total order over transactions is required (still, SPECTRE will work, but will provide a lower throughput for transactions that must be ordered).

Additional analysis on the protocol is required from the perspective of incentives. While we suggested to reward all miners of blocks with block reward, which implies selfish miners would not be able to reduce the block rewards of others, it still remains to be seen how well SPECTRE performs when miners try to selfishly maximize transaction rewards, even at the expense of others (some work in this direction was done in [13]).

---

<sup>15</sup> $\epsilon$  here simply represents a value greater than  $f_{pre\_mine} + f_{pre\_pub} + f_{post\_pub} + f_{post\_mine}$ .

## References

- [1] Kenneth J Arrow, Amartya Sen, and Kotaro Suzumura. *Handbook of Social Choice & Welfare*, volume 2. Elsevier, 2010.
- [2] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, page 13. ACM, 2016.
- [3] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *13th IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy, September 2013*.
- [4] John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.
- [5] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.
- [6] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.
- [7] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.
- [8] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in bitcoin. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 692–705. ACM, 2015.
- [9] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoins peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.
- [10] Ari Juels and John G Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, volume 99, pages 151–165, 1999.
- [11] Aggelos Kiayias and Giorgos Panagiotakos. On trees, chains and fast transactions in the blockchain. Cryptology ePrint Archive, Report 2016/545, 2016.
- [12] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 279–296, 2016.

- [13] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.
- [14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
- [15] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR Cryptology ePrint Archive*, 2016:454, 2016.
- [16] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. *Cryptology ePrint Archive*, Report 2016/917, 2016.
- [17] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.
- [18] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.
- [19] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

## 10 Appendix: The Complete Proof

### 10.1 Basic properties

**Lemma 18.** *The genesis’s vote is the final vote:  $vote(virtual(G)) = vote(genesis, G)$ .*

*Proof.* Suffice it show that if  $vote_{x,y}(genesis, G) \geq 0$  then  $vote_{x,y}(virtual(G)) \geq 0$ . If  $(x, y)$  are related topologically then by Lemma 9 all votes agree unanimously on their ordering, and in particular  $vote_{x,y}(virtual(G)) = vote_{x,y}(genesis, G) \geq 0$ . Otherwise, it cannot be the case that  $x$  or  $y$  are the *genesis* block, hence the *genesis* is a weak voter, and by line 14 we obtain

$$vote_{x,y}(virtual(G)) = \widetilde{sgn} \left( \sum_{z \in G} vote_{x,y}(z, G) \right) = \quad (3)$$

$$\widetilde{sgn} \left( vote_{x,y}(genesis, G) + \sum_{z \in future(genesis, G)} vote_{x,y}(z, G) \right) \geq \quad (4)$$

$$\widetilde{sgn} \left( \sum_{z \in future(genesis, G)} vote_{x,y}(z, G) \right) = vote_{x,y}(genesis, G) \geq 0, \quad (5)$$

hence  $vote_{x,y}(virtual(G)) \geq 0$ . □

## 10.2 Proof of Consistency

*Proof. Part I:* We first prove that for any DAG  $G$ , and for any  $tx_1, tx_2 \in T$ : if  $tx_2 \in \text{inputs}(tx_1)$  and  $[tx_1] \cap \text{GetAccepted}(G) \neq \emptyset$  then  $[tx_2] \cap \text{GetAccepted}(G) \neq \emptyset$ .

Assume  $tx_2 \in \text{inputs}(tx_1)$  and  $[tx_1] \cap \text{GetAccepted}(G, G) \neq \emptyset$  and let  $tx_1 \in [tx_1] \cap \text{GetAccepted}(G, G)$ . Consider the iteration of the second loop (line 3) over  $tx = tx_1$ . As  $tx \in \text{GetAccepted}(G, G)$  it must be the case that during this iteration the algorithm has reached line 13. This means that for any  $[tx_3] \in \text{inputs}(tx_1)$  it hasn't visited line 12; in particular for  $[tx_3] = [tx_2]$ , the condition  $[tx_2] \cap \text{GetAccepted}(G, \text{past}(z_1)) = \emptyset$  has failed, i.e.,  $[tx_2] \cap \text{GetAccepted}(G, \text{past}(z_1)) \neq \emptyset$ . To see that  $\text{GetAccepted}(G, \text{past}(z_1)) \subseteq \text{GetAccepted}(G, G)$  observe that (i) during the run of the algorithm no transaction is ever removed from  $TX$ , and that (ii) for any  $z_1 \in G \cap \text{sub}G$ , the operations (in lines 3-13) of  $\text{GetAccepted}(G, \text{sub}G)$  and  $\text{GetAccepted}(G, G)$  are identical; thus any addition of a transaction in line 13 in  $\text{GetAccepted}(G, \text{sub}G)$  occurs in  $\text{GetAccepted}(G, G)$  as well. In particular,  $[tx_2] \cap \text{GetAccepted}(G, G) \neq \emptyset$ .

*Part II:* We now prove that for any DAG  $G$ , and for any  $tx_1, tx_2 \in T$ : if  $tx_2 \in \text{conflict}(tx_1)$  and  $[tx_1] \cap \text{GetAccepted}(G, G) \neq \emptyset$  then  $[tx_2] \cap \text{GetAccepted}(G) = \emptyset$ .

Assume that  $tx_2 \in \text{conflict}(tx_1)$  and  $[tx_1] \cap \text{GetAccepted}(G, G) \neq \emptyset$  and let  $tx_1$  be an element in the latter intersection. Assume by way of negation that there exists a  $tx_2 \in [tx_2] \cap \text{GetAccepted}(G, G)$ . Then during the iteration of the first loop (line 2) over some instantiation  $z_1^1$  of  $z_1$  such that  $tx_1 \in z_1^1$ , and of the second loop (line 3) over  $tx_1$ , the algorithm has reached line 13. In particular, it did not reach line 9, hence  $z_1^2 \notin \text{past}(z_1^1)$ . For the symmetrical argument,  $z_1^1 \notin \text{past}(z_1^2)$ , which implies that  $z_1^2 \in \text{anticone}(z_1^1, G)$  (and  $z_1^1 \in \text{anticone}(z_1^2, G)$ ). Now, either  $\text{vote}_{z_1^1, z_1^2}(\text{virtual}(G)) \geq 0$  or  $\text{vote}_{z_1^2, z_1^1}(\text{virtual}(G)) \geq 0$ . Either way, line 7 was reached by either the run on  $tx_1$  or the run on  $tx_2$ , which contradicts the assumption that both runs reached line 13.  $\square$

## 10.3 Proof of Safety (blocks)

In order to simplify the analysis, we need to make some worst case assumptions regarding the behaviour of the attacker, namely, that it publishes all of its blocks immediately after time  $t$  (which represents the time at which some honest node accepted the transaction), and that before  $\text{time}(x)$  its blocks point at all available blocks. These assumptions essentially modify the DAG (in case the attacker does not carry out the optimal attack scheme). We need to prove that these modifications indeed represent the worst case. To this end we use the notion of a *pseudo-vote*. A pseudo-vote begins by first explicitly defining and *fixing* the pseudo-vote of some blocks, which we call the *initial pseudo-voters*. Then we define the pseudo-vote of the rest of the blocks as in Algorithm 1. In more detail, we replace in Algorithm 1 the  $\text{vote}()$  notation by the  $\text{p\_vote}()$  notation, and whenever the algorithm references  $\text{p\_vote}(c)$  of an initial pseudo-voter  $c$ , we refer to its fixed predetermined value. Thus, the pseudo-vote of an initial pseudo-voter might change the pseudo-vote of other blocks.

**Lemma 19.** Let  $x, y \in G = (C, E)$  such that  $G_t^v \subseteq G$ . Let  $G' = (C, E')$  be the DAG resulting from adding the following edges to  $E$ :

1.  $\forall z_1 \in G \cap \text{before}(\text{time}(x)) \cap \text{attacker}, \forall z_2 \in G \cap \text{before}(\text{time}(z_1)) \setminus \{z_1\}$ : add  $(z_1, z_2)$  to  $E$ .
2.  $\forall z_1 \in G \cap \text{attacker} \setminus G_t^v, \forall z_2 \in G_{\text{publication}(z_1), \infty}^{\text{oracle}} \cap \text{honest}$ : add  $(z_2, z_1)$  to  $E$ .

Let  $p\_vote()$  be defined by specifying the following initial pseudo-voters (and their votes):

3.  $\forall z \in (G \cap \text{attacker} \setminus G_t^v) \cup G \cap \text{before}(\text{time}(x)) \cap \text{attacker}$ :  $p\_vote_{x,y}(z, G) = +1$ .

Then  $vote_{x,y}(\text{virtual}(G), G) \leq p\_vote_{x,y}(\text{virtual}(G'), G')$ .<sup>16</sup>

Importantly, we assume here that blocks in  $G \cap \text{attacker}$  break ties in favour of  $y \prec x$ .

*Proof. Part I:* Assume by way of negation that  $vote_{x,y}(\text{virtual}(G), G) = +1$  yet nonetheless  $p\_vote_{x,y}(\text{virtual}(G), G') = -1$  (observe that this is the only case in which the claim can fail, by definition, as a virtual vote cannot take the value of 0).

Let  $b$  be a block in  $\text{future}(x, G) \cup \{\text{virtual}(G)\}$  such that  $p\_vote_{x,y}(b, G') = -1$ .  $b$  cannot belong to  $G \setminus G_t^v$  or to  $G \cap \text{before}(\text{time}(x)) \cap \text{attacker}$ , because blocks in these sets have a pseudo-vote of  $+1$ . Let  $z$  be a block in  $\text{past}(b, G)$ . Since  $b \notin (G \setminus G_t^v) \cup (G \cap \text{before}(\text{time}(x)) \cap \text{attacker})$ , there exists in  $G'$  a path from  $b$  to  $z$  that passes through an edge  $(z_2, z_1)$  satisfying the conditions of the second modification to  $G$ , and through an edge  $(z'_1, z'_2)$  satisfying those of the first modification. In particular,  $\text{time}(z_2) \geq \text{publication}(z_1) \geq t - d$ , and  $\text{time}(x) \geq \text{time}(z_2) \geq \text{time}(z'_1)$ . As  $b$  and  $z$  are the end-vertices of this path,  $\text{time}(b) \geq \text{time}(z_2) \geq t - d \geq \text{publication}(x) + d \geq \text{time}(x) + d \geq \text{time}(z'_1) + d \geq \text{time}(z) + d$ . Since  $z_2$  is honest,  $z \in \text{past}(z_2, G)$ , hence  $z \in \text{past}(b, G)$ . Combined with  $E \subset E'$  we obtain:  $\text{past}(b, G') = \text{past}(b, G)$ .

**Part II:** Let  $b$  be the earliest block in  $\text{future}(x, G) \cup \{\text{virtual}(G)\}$  for which  $vote_{x,y}(b, G) = +1$  but  $p\_vote_{x,y}(b, G') = -1$ , and let  $z$  be the latest block in  $\text{antifuture}(x, G')$  for which  $vote_{x,y}(z, \text{past}(b, G)) > p\_vote_{x,y}(z, \text{past}(b, G'))$ . If such a  $z$  exists then, similarly to the previous part, we know that  $\text{past}(z, G') = \text{past}(z, G)$ ; this proves that  $z$  is a weak voter both in  $G$  and in  $G'$ , hence that its pseudo-vote is the sign of the sum of pseudo-votes in its future.<sup>17</sup>

To see that such a  $z$  indeed exists, observe that the *genesis* satisfies these conditions: By Lemma 11  $vote_{x,y}(b, G) = vote_{x,y}(\text{virtual}(\text{past}(b, G))) = +1$  implies  $vote_{x,y}(\text{genesis}, \text{past}(b, G)) \geq 0$ , and in a similar way  $p\_vote_{x,y}(b, G') =$

<sup>16</sup>Note that  $\text{virtual}(G) = \text{virtual}(G')$ , as they share the same vertex-set.

<sup>17</sup>It cannot be the case that  $y \in \text{past}(z, G')$ , because we know that  $z$  is not an initial pseudo-voter (as its pseudo-vote is  $-1$ ), and therefore the pseudo-vote procedure would have assigned its pseudo-vote to be  $+1$ , because  $y$  is in its past but  $x$  is not in its past, in the same way the ordinary procedure does.



$p\_vote_{x,y}(virtual(past(b, G'))) = -1$  implies that  $p\_vote_{x,y}(genesis, past(b, G')) = -1$ .<sup>18</sup>

**Part III:** By the choice of  $z$ , if  $z' \in future(z, past(b, G'))$  is weak with respect to  $(x, y)$  then  $vote_{x,y}(z', past(b, G)) \leq p\_vote_{x,y}(z', past(b, G'))$ . Moreover, by the choice of  $b$ , if  $z' \in past(b, G')$  is strong w.r.t.  $(x, y)$  and  $vote_{x,y}(z', past(b, G)) = +1$  then  $p\_vote_{x,y}(z', past(b, G')) = +1$ . All in all, we have that for all  $z' \in future(z, past(b, G'))$ ,  $vote_{x,y}(z', past(b, G)) \leq p\_vote_{x,y}(z', past(b, G'))$ . Therefore:

$$\sum_{z' \in future(z, past(b, G))} vote_{x,y}(z', past(b, G)) \leq \quad (6)$$

$$\sum_{z' \in future(z, past(b, G))} p\_vote_{x,y}(z', past(b, G')) \leq \quad (7)$$

$$\sum_{z' \in future(z, past(b, G))} p\_vote_{x,y}(z', past(b, G')) + \sum_{z' \in future(z, past(b, G') \setminus past(b, G))} p\_vote_{x,y}(z', past(b, G')) = \quad (8)$$

$$\sum_{z' \in future(z, past(b, G'))} p\_vote_{x,y}(z', past(b, G')). \quad (9)$$

The last equality follows from  $future(z, past(b, G)) \subseteq future(z, past(b, G'))$ , which holds because  $E \subseteq E'$ . The inequality in (7) holds because if some  $z'$  has been added (by transforming  $G$  into  $G'$ ) to the future of some honest block, then  $z'$  must belong to the attacker, hence  $p\_vote_{x,y}(z', past(b, G')) = +1 > 0$ .

**Part IV:** Consequently, since  $z$  is a weak voter with respect to  $(x, y)$ , (6)-(9) imply that  $vote_{x,y}(z, past(b, G)) \leq p\_vote_{x,y}(z, past(b, G'))$ , which contradicts the choice of  $z$ .  $\square$

**Lemma 12.** *Let  $t \geq publication(x) + 2 \cdot d$ . Let  $z_{late}$  be the latest block in  $\overline{past}_h(x)$ . Denote:*

- $h := |anticone_h(z_{late}, G_t^{oracle})|$
- $j := |future_h(z_{late}, G_t^{oracle}) \setminus future_h(x, G_t^v)|$
- $m := |future_a(z_{late}, G_t^{oracle}) \setminus future_a(x, G_t^v)|$
- $k_1 := |G_{[t-d, t]}^{oracle} \cap honest|$
- $l := \max_{z \in G_t^{oracle} \cap honest} \left\{ \left| future_a(z, G_{time(z_{late})}^u) \right| - \left| future_h(z, G_{time(z_{late})}^u) \right| \right\}$
- $g := \sum_{z \in \overline{future}(x, G_t^v)} vote_{y,x}(z, G_t^v)$

<sup>18</sup>The equality  $p\_vote_{x,y}(b, G') = p\_vote_{x,y}(virtual(past(b, G')))$  holds because  $b$  is either a strong voter w.r.t.  $(x, y)$  or the virtual voter.

Then,

$$\begin{aligned} \widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathbb{G}} &\subseteq \left\{ \exists s \geq t, \exists u \in \text{honest s.t. } \left| G_{[t,s]}^u \cap \text{attacker} \right| \right. \\ &\geq \left. \left| G_{[t,s]}^u \cap \text{honest} \right| + g - 2 \cdot h - j - k_1 - l - m \right\}. \end{aligned} \quad (10)$$

*Proof. Part I:* In the proof below we make the following assumption: Any attacker-block  $z$  created before  $\text{time}(x)$  always votes in favour of  $y \prec x$  (even if it is supposed to vote otherwise according to Algorithm 1). We further assume that any such  $z$  satisfies  $\overline{\text{past}}(z) = G_{\text{time}(z)}^{\text{oracle}}$ , i.e., it points at all blocks available at the time of its creation. Finally, we assume that the attacker releases all of his blocks to all nodes in  $\text{honest} \setminus \{v\}$  precisely at time  $t$  and onward. The previous lemma implies that these are indeed worst case assumptions: Take  $G$  to be any  $G_s^u$ . Then, what the lemma shows is that as long as  $p\text{-vote}_{x,y}(\text{virtual}(G_s^u)) = -1$ , also  $\text{vote}_{x,y}(\text{virtual}(G_s^u)) = -1$  (under the worst case assumption that ties are always broken in favour of  $y$ ).<sup>19</sup> The analysis below applies, formally, to  $p\text{-vote}()$  as formalized in the previous lemma (specifically in (3)). Nevertheless, now that the argument has been formally made, we omit this notation henceforth.

**Part II:** Let us look at the following chain of implications:

$$\begin{aligned} \text{vote}_{x,y}(\text{virtual}(G_s^u)) \geq 0 &\Rightarrow \text{vote}_{x,y}(\text{genesis}, G_s^u) \geq 0 \Rightarrow \\ &\sum_{z' \in \text{future}(\text{genesis}, G_s^u)} \text{vote}_{x,y}(z', G_s^u) \geq 0 \end{aligned} \quad (11)$$

The first implication follows from Lemma 11. The second one follows from the definition of  $\text{genesis}$ 's vote.<sup>20</sup> Thus,

$$\begin{aligned} \widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathbb{G}} &= \cup_{u \in \text{honest}, s \in [t, \infty)} \widehat{\mathcal{E}}_s^u(x, y)^{\mathbb{G}} = \\ &\{ \exists u \in \text{honest}, \exists s \geq t : \text{vote}_{x,y}(\text{virtual}(G_s^u)) \geq 0 \} \end{aligned}$$

However, if there exists such an  $s$  as the latter event requires, then we can look at the first such  $s$ . With respect to it, between  $t$  and  $s$  all honest votes were in favour of  $x \prec y$ ; this is because for any  $\text{honest}$  block  $z'$  with  $\text{time}(z') \in [t, s)$ ,  $\text{past}(z') = G_{\text{time}(z')}^{\text{node}(z')}$ , hence  $\text{vote}(z') = \text{vote}(\text{virtual}(G_{\text{time}(z')}^{\text{node}(z')}))$ , and by the choice of  $s$  as the earliest time for which an honest node's DAG's virtual block votes in favour of  $y \preceq x$ , we know that  $\text{vote}_{x,y}(z', G_s^u) = -1$ .

**Part III:** Below, the notation  $G_{[t_1, t_2]}^u$  stands for  $G_s^u \cap \text{before}(t_2) \setminus \text{before}(t_1)$ .

<sup>19</sup>In fact, we use the lemma with a slight modification: The second modification does not apply to all such  $(z_2, z_1)$  satisfying the specified conditions, rather to a subset thereof, since blocks created by node  $v$  between  $t$  and  $t + d$  need not point at all attacker blocks in  $G \setminus G_s^v$ . It is easy to see, however, that the proof of the lemma remains intact (and it remains so when applying the second modification to *any* subset of  $(G \cap \text{attacker} \setminus G_t^v) \times (G_{[\text{publication}(z_1), \infty)}^{\text{oracle}} \cap \text{honest})$ ).

<sup>20</sup>Here we implicitly assume that  $x$  and  $y$  are not related topologically, which rules out the option that  $x = \text{genesis}$  or  $y = \text{genesis}$ , hence  $\text{genesis}$  is weak w.r.t.  $(x, y)$ . If they are related topologically, the result is trivial, for all votes are then forever unanimous in the same direction (Lemma 9).

We claim that for all  $z \in \overline{\text{past}_h}(x)$ :

$$\begin{aligned} \text{vote}_{x,y}(z, G_s^u) &\leq \widehat{\text{sgn}} \left( \left| G_{[t,s]}^u \cap \text{attacker} \right| - \left| G_{[t,s]}^u \cap \text{honest} \right| \right) \\ &+ 2 \cdot h + l + k_1 + j + m - g. \end{aligned}$$

We prove the claim by a complete induction on  $D(z) := |\text{future}(z, \overline{\text{past}_h}(x))|$ . Assume we have proved the claim for any  $z$  with  $D(z) < D$ . We now prove it for  $z$  with  $D(z) = D$ . If  $z = x$  then  $\text{vote}_{x,y}(z, G_s^u) = -1$  hence the above inequality is satisfied trivially. Otherwise,  $z$  is a weak voter, and  $\text{vote}_{x,y}(z, G_s^u)$  is given by the sign of the sum of votes in its future. We decompose these voters into three subsets: members of  $\text{future}(z, G_{\text{time}(z_{\text{late}})}^u)$ , members of  $\text{future}(z, G_{[\text{time}(z_{\text{late}}), t]}^u)$ , and members of  $\text{future}(z, G_{[t,s]}^u)$ .

1. Members of  $\text{future}(z, G_{\text{time}(z_{\text{late}})}^u)$ : By the induction hypothesis we know that all blocks in  $\text{future}_h(z, \overline{\text{past}_h}(z_{\text{late}}))$  vote in favour of  $x \prec y$ , and by the choice of  $z_{\text{late}}$  we have that  $\text{future}_h(z, G_{\text{time}(z_{\text{late}})}^u) \setminus \overline{\text{past}}(z_{\text{late}}) = \text{anticone}_h(z, G_{\text{time}(z_{\text{late}})}^u)$ . Thus,  $\sum_{z' \in \text{future}_h(z, G_{\text{time}(z_{\text{late}})}^u)} \text{vote}_{x,y}(z', G_s^u) \leq 2 \cdot \left| \text{anticone}_h(z_{\text{late}}, G_{\text{time}(z_{\text{late}})}^u) \right| - \left| \text{future}_h(z, G_{\text{time}(z_{\text{late}})}^u) \right|$ . We obtain:

$$\begin{aligned} &\sum_{z' \in \text{future}(z, G_{\text{time}(z_{\text{late}})}^u)} \text{vote}_{x,y}(z', G_s^u) \leq \\ &2 \cdot \left| \text{anticone}_h(z_{\text{late}}, G_{\text{time}(z_{\text{late}})}^u) \right| \\ &- \left| \text{future}_h(z, G_{\text{time}(z_{\text{late}})}^u) \right| + \left| \text{future}_a(z, G_{\text{time}(z_{\text{late}})}^u) \right|. \end{aligned}$$

2. Members of  $\text{future}(z, G_{[\text{time}(z_{\text{late}}), t]}^u)$ :

- (a) Honest blocks: By *Part I* we have that  $\text{future}_a(z, G_{[\text{time}(z_{\text{late}}), t]}^u) \setminus \text{future}_a(z_{\text{late}}, G_{[\text{time}(z_{\text{late}}), t]}^u) = \emptyset$ . This implies that  $\text{anticone}_h(z_{\text{late}}, G_{[\text{time}(z_{\text{late}}), \text{time}(z_{\text{late}})+d]}^u) \supseteq \text{future}(z, G_{[\text{time}(z_{\text{late}}), t]}^u) \setminus$

$future(z_{late}, G_t^u)$ . We obtain:

$$\begin{aligned}
& \sum_{z' \in future_h(z, G_{[time(z_{late}), t]}^u)} vote_{x,y}(z', G_s^u) \leq \\
& \sum_{z' \in future_h(z_{late}, G_t^u)} vote_{x,y}(z', G_s^u) + \\
& \left| anticone_h(z_{late}, G_{[time(z_{late}), time(z_{late})+d]}^u) \right| \leq \\
& \sum_{z' \in future_h(z_{late}, G_t^v)} vote_{x,y}(z', G_s^u) + \\
& \left| anticone_h(z_{late}, G_{[time(z_{late}), time(z_{late})+d]}^u) \right| + \\
& \left| G_{[t-d, t]}^{oracle} \cap honest \right| \leq \\
& \sum_{z' \in future_h(x, G_t^v)} vote_{x,y}(z', G_s^u) + \\
& \left| anticone_h(z_{late}, G_{[time(z_{late}), time(z_{late})+d]}^u) \right| + \\
& \left| G_{[t-d, t]}^{oracle} \cap honest \right| + \\
& |future_h(z_{late}, G_t^v) \setminus future_h(x, G_t^v)|.
\end{aligned}$$

(b) Attacker blocks: We utilize our worst case assumptions described in *Part I* to obtain:

$$\begin{aligned}
& \sum_{z' \in future_a(z, G_{[time(z_{late}), t]}^u)} vote_{x,y}(z', G_s^u) = \\
& \sum_{z' \in future_a(z_{late}, G_{[time(z_{late}), t]}^u)} vote_{x,y}(z', G_s^u) \leq \\
& \sum_{z' \in future_a(x, G_t^v)} vote_{x,y}(z', G_s^u) + \\
& |future_a(z_{late}, G_t^u) \setminus future_a(x, G_t^v)|.
\end{aligned}$$

(c) All blocks: We combine the honest and attacker blocks in

$future(z, G_{[time(z_{late}), t]}^u)$  to obtain:

$$\begin{aligned}
& \sum_{z' \in future(z, G_{[time(z_{late}), t]}^u)} vote_{x,y}(z', G_s^u) \leq \\
& \sum_{z' \in future_h(x, G_t^v)} vote_{x,y}(z', G_s^u) + \\
& \left| anticone_h(z_{late}, G_{[time(z_{late}), time(z_{late})+d]}^u) \right| + \\
& \left| G_{[t-d, t]}^{oracle} \cap honest \right| + |future_h(z_{late}, G_t^v) \setminus future_h(x, G_t^v)| \\
& + \sum_{z' \in future_a(x, G_t^v)} vote_{x,y}(z', G_s^u) + \\
& |future_a(z_{late}, G_t^v) \setminus future_a(x, G_t^v)| = \tag{12}
\end{aligned}$$

$$\begin{aligned}
& g + \left| anticone_h(z_{late}, G_{[time(z_{late}), time(z_{late})+d]}^u) \right| + \tag{13} \\
& \left| G_{[t-d, t]}^{oracle} \cap honest \right| + |future_h(z_{late}, G_t^v) \setminus future_h(x, G_t^v)| \\
& + |future_a(z_{late}, G_t^v) \setminus future_a(x, G_t^v)|.
\end{aligned}$$

3. Members of  $future(z, G_{[t, s]}^u)$ : Finally, by the choice of  $s$ , all honest blocks created between  $t$  and  $s$  vote in favour of  $x \prec y$ , hence

$$\begin{aligned}
& \sum_{z' \in future(z, G_{[t, s]}^u)} vote_{x,y}(z', G_s^u) \leq \\
& - \left| future_h(z, G_{[t, s]}^u) \right| + \left| future_a(z, G_{[t, s]}^u) \right| \leq \\
& - \left| G_{[t, s]}^u \cap honest \right| + \left| G_{[t, s]}^u \cap attacker \right|,
\end{aligned}$$

where we used again the fact that  $t \geq publication(x) + d \geq publication(z) + d$ .

4. Combining all the above results we obtain:

$$\sum_{z' \in \text{future}(z, G_s^u)} \text{vote}_{x,y}(z', G_s^u) \leq \quad (14)$$

$$2 \cdot \left| \text{anticone}_h \left( z_{\text{late}}, G_{\text{time}(z_{\text{late}})}^u \right) \right| - \left| \text{future}_h \left( z, G_{\text{time}(z_{\text{late}})}^u \right) \right| + \left| \text{future}_a \left( z, G_{\text{time}(z_{\text{late}})}^u \right) \right| \quad (15)$$

$$+ g + \left| \text{anticone}_h \left( z_{\text{late}}, G_{[\text{time}(z_{\text{late}}), \text{time}(z_{\text{late}})+d]}^u \right) \right| + \left| G_{[t-d, t]}^{\text{oracle}} \cap \text{honest} \right| + \left| \text{future}_h(z_{\text{late}}, G_t^v) \setminus \text{future}_h(x, G_t^v) \right| + \left| \text{future}_a(z_{\text{late}}, G_t^u) \setminus \text{future}_a(x, G_t^v) \right| + \quad (16)$$

$$- \left| G_{[t, s]}^u \cap \text{honest} \right| + \left| G_{[t, s]}^u \cap \text{attacker} \right| \leq \quad (17)$$

$$(18)$$

$$2 \cdot h + l + k_1 - g + \left| G_{[t, s]}^u \cap \text{attacker} \right| - \left| G_{[t, s]}^u \cap \text{honest} \right| + \left| \text{future}_h(z_{\text{late}}, G_t^v) \setminus \text{future}_h(x, G_t^v) \right| + \left| \text{future}_a(z_{\text{late}}, G_t^u) \setminus \text{future}_a(x, G_t^v) \right| \leq \quad (19)$$

$$2 \cdot h + l + k_1 - g + \left| G_{[t, s]}^u \cap \text{attacker} \right| - \left| G_{[t, s]}^u \cap \text{honest} \right| + \left| \text{future}_h(z_{\text{late}}, G_t^{\text{oracle}}) \setminus \text{future}_h(x, G_t^v) \right| + \left| \text{future}_a(z_{\text{late}}, G_t^{\text{oracle}}) \setminus \text{future}_a(x, G_t^v) \right| = \quad (20)$$

$$2 \cdot h + l + k_1 - g + j + m + \left| G_{[t, s]}^u \cap \text{attacker} \right| - \left| G_{[t, s]}^u \cap \text{honest} \right|. \quad (21)$$

As  $z$  is a weak voter, we conclude that  $\text{vote}_{x,y}(z, G_s^u) \leq \widehat{\text{sgn}} \left( \left| G_{[t, s]}^u \cap \text{attacker} \right| - \left| G_{[t, s]}^u \cap \text{honest} \right| + 2 \cdot h + l + k_1 + g + j + m \right)$ .

**Part IV:** In particular, for  $z = \text{genesis}$ , the event  $\text{vote}_{x,y}(\text{genesis}, G_s^u) \geq 0$  is contained in the event where  $\left| G_{[t, s]}^u \cap \text{attacker} \right| \geq \left| G_{[t, s]}^u \cap \text{honest} \right| - 2 \cdot h - l - k_1 - g - j - m$ . By (11), this event contains also  $\widehat{\mathcal{E}}_u^s(x, y)^{\mathfrak{G}}$ , for all  $u \in \text{honest}$  and  $s \geq t$ , hence it contains also their union  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{\text{all}}(x, y)^{\mathfrak{G}}$ .  $\square$

**Lemma 13.** *Given the parameters of Lemma 12,*

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^c\right) \leq \sum_{h'=0}^{\infty} \mathcal{P}_{oiiss}(d \cdot \alpha \cdot \lambda, h') \cdot \left(\frac{\alpha}{1-\alpha}\right)^{(g-2 \cdot h-j-k_1-l-m-h')^+}. \quad (22)$$

*Proof.* Since all nodes  $u \in honest$  receive honest blocks with a delay of  $d$  seconds at most, we have that  $\left|G_{[t,s]}^u \cap attacker\right| - \left|G_{[t,s]}^u \cap honest\right| \leq \left|future_a(z_{late}, G_{[t,s]}^{oracle})\right| - \left|future_h(z_{late}, G_{[t, \max\{s-d, t\}]}^{oracle})\right|$ . We further upper bound  $\left|future_a(x, G_{[s',s]}^{oracle})\right|$  by  $\left|future_a(x, G_{[s-d,s]}^{oracle})\right|$ , and observe that the latter follows a Poisson distribution with parameter  $\alpha \cdot d \cdot \lambda$ ; we denote this variable by  $h'$ . For any given value of  $h'$ , the variable  $\left|future_a(x, G_{s'}^{oracle} \setminus G_t^{oracle})\right| - \left|future_h(x, G_{s'}^{oracle} \setminus G_t^{oracle})\right| + h'$  can be modeled as a random walk  $X_i$  (where the  $i$ th step is the creation-time of the  $i$ th block after time  $t$ ), with  $X_0 = h'$ , and with a drift of  $\alpha$  towards positive infinity. The probability that  $X_i$  would ever reach the interval  $[-h-j-k_1-h'-l-m+g, +\infty)$  is  $\left(\frac{\alpha}{1-\alpha}\right)^{g-2 \cdot h-j-k_1-l-m-h'}$ , if  $g > h+j+k_1+l+m+h'$ , and 1 otherwise (see [19, 17]).  $\square$

**Corollary 14.** *If*

- $j \geq |anticone_h(x, G_t^{oracle})|$
- $l \geq \max_{z \in G_t^{oracle} \cap honest} \left\{A_{time(x)}^z - H_{time(x)}^z\right\}$
- $n_x \geq future_h(x, G_t^{oracle})$
- $g \leq \sum_{z \in \overline{future}(x, G_t^v)} vote_{y,x}(z, G_t^v)$ .

*Then*

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)\right) \leq \sum_{k=0}^{\infty} \mathcal{P}_{oiiss}((2-\alpha) \cdot d \cdot \lambda, k) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiiss}(d \cdot (1-\alpha) \cdot \lambda, h) \cdot \sum_{m=0}^{\infty} \binom{n_x + j + h + m - 1}{m} \cdot (1-\alpha)^{n_x + j + h} \cdot \alpha^m \cdot \left(\frac{\alpha}{1-\alpha}\right)^{(g-2 \cdot h-k-j-l-m)^+}. \quad (23)$$

*Proof.* We build on the results of previous lemmas. The proof of Lemma 12, which is deterministic, remains intact when the corresponding parameters serve as bounds; see (14)-(21).

The variables  $k_1$ ,  $\left| \overline{anticone}_h \left( z_{late}, G_{[time(z_{late}), time(z_{late})+d]}^u \right) \right|$ , and  $h'$  are the sum of independent Poisson processes; the parameter of the first two is  $d \cdot (1 - \alpha) \cdot \lambda$ , and the parameter of  $h'$  is  $d \cdot \alpha \cdot \lambda$ . Thus, their sum is a new Poisson variable  $k$  with parameter  $(2 \cdot (1 - \alpha) + \alpha) \cdot d \cdot \lambda = (2 - \alpha) \cdot d \cdot \lambda$ . The variable  $\left| \overline{anticone}_h \left( z_{late}, G_{time(z_{late})}^u \right) \right|$  is an additional Poisson variable with parameter  $d \cdot (1 - \alpha) \cdot \lambda$ . We denote it by  $h$  (thereby overriding its original meaning in Lemma 12).

Lemma 12 uses the variable  $m = \left| \overline{future}_a(z_{late}, G_t^{oracle}) \setminus \overline{future}_a(x, G_t^v) \right|$  which is upper bounded by  $\left| \overline{future}_a(z_{late}, G_t^{oracle}) \right|$ . Provided that the honest network has created precisely  $n$  blocks since the creation of  $z_{late}$ , the number of blocks created by the attacker at the same time follows a negative binomial distribution (see [17]), i.e., it takes the value  $m$  with probability  $\binom{n+m-1}{m} \cdot (1 - \alpha)^n \cdot \alpha^m$ . In the worst case, all of these blocks belong to  $\overline{future}(z_{late}, G_t^{oracle})$ . Here, again, it is sufficient to upper bound  $n$ , since increasing the parameter  $n$  results in a distribution over  $m$  that stochastically dominates (in first order) the original one. The number of honest blocks created after  $time(z_{late})$  (up to time  $t$ ) is upper bounded by  $\overline{antipast}_h(z_{late}, G_t^{oracle})$ , since blocks in  $\overline{past}_h(z_{late})$ . We thus have:  $n \leq \left| \overline{anticone}_h(z_{late}, G_t^v) \right| + \left| \overline{future}_h(z_{late}, G_t^v) \setminus \overline{future}_h(x, G_t^v) \right| + \left| \overline{future}_h(x, G_t^v) \right| \leq h + j + n_x$ .

Finally, as  $l$  and  $j$  are upper bounds and  $g$  is a lower bound to the corresponding variables from Lemma 12, one could simply turn all equalities in its proof (and in the proof of Lemma 13) into “ $\leq$ ” inequalities and the proof remains intact.  $\square$

Below we revisit previous results, regarding the case where  $x$  is known to be an honest block, and to the case where one needs to defend a group of blocks rather than an individual block.

**Lemma 20.** *Assume that  $node(x) \in honest$  and that  $publication(y) \geq publication(x) + d$ . Let  $z_{late}$  be the latest block in  $\overline{anticone}_h(x, G_t^v)$  and let  $z_{early}$  be the earliest block in  $\overline{anticone}_h(x, G_t^v)$ . Furthermore, assume:*

- $l := \max_{z \in \overline{future}_a \cap honest} \left\{ \left| \overline{future}_a \left( z, G_{time(z_{early})}^u \right) \right| - \left| \overline{future}_h \left( z, G_{time(z_{early})}^u \right) \right| \right\}$
- $n_x \geq \max_{x' \in \overline{anticone}_h(x)} \left\{ \left| \overline{future}_h \left( x', G_t^{oracle} \right) \right| \right\}$
- $g \leq \max_{x' \in \overline{anticone}_h(x, G_t^v)} \left\{ z \in \overline{future}(x', G_t^v) : vote_{y,x}(z, G_t^v) = -1 \right\} - \min_{x' \in \overline{anticone}_h(x, G_t^v)} \left\{ z \in \overline{future}(x', G_t^v) : vote_{y,x}(z, G_t^v) = +1 \right\}$ .



Then,

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathcal{G}}\right) \leq \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot \lambda, h). \quad (24)$$

$$\sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m. \quad (25)$$

$$\left(\frac{\alpha}{1 - \alpha}\right)^{(g-h-l-m)^+}.$$

*Proof.* Let  $k_1 \geq |G_{[t-d, t]}^{oracle} \cap honest|$ , and let  $m := |future_a(z_{early}, G_t^{oracle}) \setminus future_a(z_{late}, G_t^v)|$ . We adjust the analysis from the proof of Lemma 12. We claim that for all  $z \in \overline{anticone}_h(x, G_t^v)$ :

$$vote_{x,y}(z, G_s^u) \leq \widehat{sgn} \left( |G_{[t,s]}^u \cap attacker| - |G_{[t,s]}^u \cap honest| \right. \\ \left. l + k_1 + m - g \right).$$

We prove the claim by a complete induction on  $D(z) := |future(z, \overline{anticone}_h(x))|$ . Assume we have proved the claim for any  $z$  with  $D(z) < D$ . We now prove it for  $z$  with  $D(z) = D$ . If  $z = x$  then  $vote_{x,y}(z, G_s^u) = -1$  hence the above inequality is satisfied trivially. Otherwise,  $z$  is a weak voter, as  $y \notin \overline{past}(z)$  by the assumption on  $publication(y)$ , therefore  $vote_{x,y}(z, G_s^u)$  is given by the sign of the sum of votes in its future. We decompose these voters into three subsets: members of  $future(z, G_{time(z_{late})}^u)$ , members of  $future(z, G_{[time(z_{late}), t]}^u)$ , and members of  $future(z, G_{[t,s]}^u)$ .

1. Members of  $future(z, G_{time(z_{late})}^u)$ : By the induction hypothesis we know that all blocks in  $future_h(z, \overline{anticone}_h(z_{late}))$  vote in favour of  $x \prec y$ , hence We obtain:

$$\sum_{z' \in future(z, G_{time(z_{late})}^u)} vote_{x,y}(z', G_s^u) \leq \\ - \left| future_h(z, G_{time(z_{late})}^u) \right| + \left| future_a(z, G_{time(z_{late})}^u) \right|.$$

2. Members of  $future(z, G_{[time(z_{late}), t]}^u)$ : Every  $z'$  in this set belongs to  $future(x')$

for some  $x' \in \overline{\text{anticone}_h}(x, G_t^v)$ , therefore, by the definition of  $g$ :

$$\begin{aligned}
& \sum_{z' \in \text{future}(z, G_{[time(z_{late}), t]}^u)} \text{vote}_{x,y}(z', G_s^u) \leq \\
& -g + \left| \text{future}_a(z_{early}, G_t^{\text{oracle}}) \setminus \text{future}_a(z_{late}, G_t^v) \right| + \\
& \left| \text{future}_h(z, G_t^u) \setminus \text{future}_h(z, G_t^v) \right| \leq \tag{26} \\
& -g + \left| \text{future}_a(z_{early}, G_t^{\text{oracle}}) \setminus \text{future}_a(z_{late}, G_t^v) \right| + \left| G_{[t-d, t]}^{\text{oracle}} \cap \text{honest} \right| = \\
& -g + m + k_1.
\end{aligned}$$

3. Members of  $\text{future}(z, G_{[t, s]}^u)$ : By the choice of  $s$ , all honest blocks created between  $t$  and  $s$  vote in favour of  $x \prec y$ , hence

$$\begin{aligned}
& \sum_{z' \in \text{future}(z, G_{[t, s]}^u)} \text{vote}_{x,y}(z', G_s^u) \leq \\
& - \left| \text{future}_h(z, G_{[t, s]}^u) \right| + \left| \text{future}_a(z, G_{[t, s]}^u) \right| \leq \\
& - \left| G_{[t, s]}^u \cap \text{honest} \right| + \left| G_{[t, s]}^u \cap \text{attacker} \right|,
\end{aligned}$$

where we used the fact that  $t \geq \text{publication}(x) + d \geq \text{publication}(z) + d$ .

4. All in all,

$$\sum_{z' \in \text{future}(z, G_s^u)} \text{vote}_{x,y}(z', G_s^u) \leq \tag{27}$$

$$l + k_1 + m - g - \left| G_{[t, s]}^u \cap \text{honest} \right| + \left| G_{[t, s]}^u \cap \text{attacker} \right| \leq \tag{28}$$

$$\begin{aligned}
& l + k_1 + m - g - \left| \text{future}_h(z_{late}, G_{[t, \max\{s-d, t\}]}^{\text{oracle}}) \right| + \\
& \left| \text{future}_a(z_{late}, G_{[t, s]}^{\text{oracle}}) \right| \tag{29}
\end{aligned}$$

Therefore, the event where for some  $s$  and some  $u$ ,  $\text{vote}_{x,y}(\text{virtual}(G_s^u)) \geq 0$  is contained in the event where (29) is non-negative. As in the proof of Lemma 13, the probability of the latter event is upper bounded by  $\left(\frac{\alpha}{1-\alpha}\right)^{(l+k_1+h'+m-g)^+}$ , where  $h'$  equals  $\left| \text{future}_a(x, G_{[s', s]}^{\text{oracle}}) \right|$ . We then combine  $k_1$  and  $h'$  into one Poisson variable

$h$  with parameter  $\alpha \cdot d \cdot \lambda + (1 - \alpha) \cdot d \cdot \lambda = d \cdot \lambda$ , to obtain:

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathfrak{G}}\right) \leq \sum_{h=0}^{\infty} \mathcal{P}_{oisss}(d \cdot \lambda, h). \quad (30)$$

$$\sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m. \quad (31)$$

$$\left(\frac{\alpha}{1 - \alpha}\right)^{(g-h-l-m)^+}.$$

□

**Corollary 21.** *If in addition to Lemma 20's assumptions we know that  $publication(y) \geq t$ , then*

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathfrak{G}}\right) \leq \sum_{h=0}^{\infty} \mathcal{P}_{oisss}(d \cdot \alpha \cdot \lambda, h). \quad (32)$$

$$\sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m. \quad (33)$$

$$\left(\frac{\alpha}{1 - \alpha}\right)^{(g-h-l-m)^+}.$$

*Proof.* Given that  $y$  wasn't published until time  $t$ , we know that all honest blocks in  $future(x, G_{[t-d, t]}^{oracle})$  vote in favour of  $x$ , hence the reduction of  $k_1 = |G_{[t-d, t]}^{oracle} \cap honest|$  in (26) is superfluous, and we thus only need to reduce  $h'$ , the Poisson variable with parameter  $d \cdot \alpha \cdot \lambda$  from Lemma 13. □

**Corollary 22.** *Let  $X \subseteq G_t^v \cap honest$  and  $Y \subseteq G_t^{oracle} \setminus G_t^v$ . Assume further that elements in  $X$  do not relate topologically to one another (i.e.,  $\forall x_1, x_2 \in X, x_1 \in anticone(x_2, G_t^v)$ ). Let  $z_{late}$  be the latest block in  $X$ , let  $z_{early}$  be the earliest block in  $X$ .*

*Then,*

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathfrak{G}}\right) \leq \sum_{h=0}^{\infty} \mathcal{P}_{oisss}((3 - 2 \cdot \alpha) \cdot d \cdot \lambda, h).$$

$$\sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{(n_x - h - l - m)^+}.$$

*Proof.* We adjust the result of Lemma 20. The main modification is that now  $g$  must relate to all pairs  $(x, y)$ . Define:  
 $g := \max_{x_1, x_2 \in X} \{z \in \overline{future}(x_1, G_t^v) : vote_{y, x_2}(z, G_t^v) = -1\} - \min_{x_1, x_2 \in X} \{z \in \overline{future}(x_1, G_t^v) : vote_{y, x_2}(z, G_t^v) = +1\}$ .  
 Observe that in the interval  $[time(z_{early}) + 2 \cdot d, t]$  all honest blocks belong to  $\cap_{x \in X} \overline{future}(x, G_t^v)$ .

In particular, if we denote  $h' := n_x - g$  we have that  $h'$  is upper bounded by a Poisson variable with parameter  $2 \cdot d \cdot \lambda$ . We then apply the analysis done in the proof of Lemma 20, with  $s$  being the first time at which for some  $(x, y) \in X \times Y$ ,  $vote_{x,y}(virtual(G_s^u)) \geq 0$ . Combining the result of that lemma with the probability distribution over  $h'$  we conclude that the probability of the event  $\cup_{(x,y) \in X \times Y} \widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$  is at most

$$\begin{aligned} \Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^c\right) &\leq \sum_{h'=0}^{\infty} \mathcal{P}_{oiss}(2 \cdot d \cdot (1 - \alpha) \cdot \lambda, h') \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot \lambda, h) \cdot \\ &\sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{(n_x - h' - h - l - m)^+} = \\ &\sum_{h=0}^{\infty} \mathcal{P}_{oiss}((3 - 2 \cdot \alpha) \cdot d \cdot \lambda, h) \cdot \\ &\sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{(n_x - h - l - m)^+}. \end{aligned}$$

□

**Corollary 15.** *If in addition to the assumptions of Corollary 14 we assume that  $M \leq |future_a(x, G_t^v)|$ , then*

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y) \mid |future_a(x, G_t^v)| \geq M\right) \leq \quad (34)$$

$$\begin{aligned} &\sum_{k=0}^{\infty} \mathcal{P}_{oiss}((2 - \alpha) \cdot d \cdot \lambda, k) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot (1 - \alpha) \cdot \lambda, h) \cdot \\ &\left(\sum_{m'=M}^{\infty} \binom{n_x + j + h + m' - 1}{m'} \cdot (1 - \alpha)^{n_x + j + h} \cdot \alpha^{m'}\right)^{-1} \cdot \\ &\sum_{m=M}^{\infty} \binom{n_x + j + h + m - 1}{m} \cdot (1 - \alpha)^{n_x + j + h} \cdot \alpha^m \cdot \\ &\left(\frac{\alpha}{1 - \alpha}\right)^{(g - 2 \cdot h - k - j - l - (m - M))^+}. \end{aligned} \quad (35)$$

We denote the RHS of this inequality by  $f_{post\_mine}(n_x, g, j, l, M)$ . We note that from Lemmas 24, 31, and 29 it follows that, in order to compute  $f_{post\_mine}$ , one can truncate these sums and suffer an exponentially low error.

*Proof.* Under the assumption on  $M$  we have  $|future_a(z_{late}, G_t^{oracle}) \setminus future_a(x, G_t^v)| = |future_a(z_{late}, G_t^{oracle})| - |future_a(x, G_t^v)| \geq |future_a(z_{late}, G_t^{oracle})| - M$ .

We then adjust the result of Corollary 14 and adjust (1) to account for the above updated definition of  $m$ . Thus, in the exponent, we substitute  $m - M$  for  $m$  and write:  $\left(\frac{\alpha}{1 - \alpha}\right)^{(g - 2 \cdot h - j - k - l - (m - M))^+}$ . Next, the updated probability distribution over  $m - M$

can be obtained by conditioning the negative binomial distribution (described in the proof of Corollary 14) on its being larger than or equal to  $M$ ; indeed, the  $M$  blocks of  $future_a(x, G_t^v)$  were created after  $z_{late}$  (and before time  $t$ ), and  $future_a(x, G_t^v) \subseteq future_a(z_{late}, G_t^{oracle})$ . Consequently, the probability distribution over  $m - M$  is given by

$$\Pr(m - M) = \left( \sum_{m'=M}^{\infty} \binom{n_x + j + h + m' - 1}{m'} \cdot (1 - \alpha)^{n_x + j + h} \cdot \alpha^{m'} \right)^{-1} \cdot \binom{n_x + j + h + m - 1}{m} \cdot (1 - \alpha)^{n_x + j + h} \cdot \alpha^m,$$

and we arrive at the desired term. The rest of the arguments in the proof of Corollary 14 remain unaffected.  $\square$

### 10.3.1 Numerical method to calculate $f_{pre\_mine}$

- Put  $\delta := \alpha \cdot \lambda \cdot d$ . Pick some  $N \gg 1$ ,<sup>21</sup> and define a matrix  $T \in \mathbb{R}_{N \times N}$  as follows. For all  $1 \leq l < N - 1$ ,  $T_{l-1,l} = 1 - \alpha$ ,  $T_{l+1,l} = \alpha$ , and for  $l = N - 1$ :  $T_{l-1,l} = 1 - \alpha$ ,  $T_{l,l} = \alpha$ . The first column of the matrix is defined by:  $T_{0,0} := (1 - \alpha) \cdot e^{-\delta}$ ,  $T_{1,0} = e^{-\delta} \cdot \alpha + e^{-\delta} \cdot \delta$ , for  $1 < l < N - 1$ :  $T_{l,0} = e^{-\delta} \cdot \frac{\delta^l}{l!}$ , and for  $l = N - 1$ :  $T_{l,0} = 1 - \sum_{l'=0}^{N-2} e^{-\delta} \cdot \frac{\delta^{l'}}{l'!}$ .
- Find the eigenvector of  $T$  corresponding to the eigenvalue 1, and denote it  $\pi$ . Define  $\Pi(l) := \sum_{l'=0}^l \pi(l')$ , and, finally, define  $f_{pre\_mine}(l) := 1 - \Pi((l - 1)^+)$ .

The matrix  $T$  is the transition probability matrix of a special reflecting random walk  $(X_k)$  over the nonnegative integers:  $T_{i,j} := \Pr(X_{k+1} = i \mid X_k = j)$ . At every position (apart from the edges 0 and  $N - 1$ ) the walk takes a step towards negative infinity w.p.  $(1 - \alpha)$  and towards positive infinity w.p.  $\alpha$ . Whenever it reaches the origin, it jumps to its next position in  $\{0, 1, \dots, N - 1\}$  according to a (modified) Poisson distribution. It is easy to see that this random walk induces an ergodic Markov chain, hence it has a unique stationary distribution, which we denoted  $\pi$ ;  $\Pi$  is the cumulative probability function of  $\pi$ .

**Lemma 23.** *For all  $\hat{r} \geq r$  and for all For all  $l \in \mathbb{N}$ :*

$$\Pr \left( \max_{z \in G_r^{oracle} \cap honest} \{ |future_a(z, G_r^{oracle})| - |future_h(z, G_r^{oracle})| \} > l \right) \leq f_{pre\_mine}(l). \quad (36)$$

<sup>21</sup>By Lemma 24, to achieve an error of at most  $\hat{\epsilon}$  it suffices to choose  $N$  such that  $\left(\frac{\alpha}{1-\alpha}\right)^{N-1} < \hat{\epsilon}/2$  and  $e^{-d \cdot \alpha \cdot \lambda} \cdot \frac{(d \cdot \alpha \cdot \lambda)^N}{N!} < \hat{\epsilon}/2$ . In particular,  $N$  is logarithmic in  $\hat{\epsilon}$ .

*Proof. Part I:* We prove the result assuming the maximum is taken over all  $z \in G_r^{oracle} \cap honest$ ; taking then the maximum over all  $z \in G_{\hat{r}}^{oracle} \cap honest$  does not change the result, because the variable  $\{|future_a(z, G_r^{oracle})| - |future_h(z, G_r^{oracle})|\}$  is nonnegative (as will be shown below), and its value for  $z \in G_{\hat{r}}^{oracle} \setminus G_r^{oracle}$  is zero.

We show that the variable  $\max_{z \in G_s^{oracle} \cap honest} \{|future_a(z, G_s^{oracle})| - |future_h(z, G_s^{oracle})|\}$  can be modeled as a reflecting random walk (with some special behaviour when the walk visits the origin, due to the honest network's inner delay  $d$ ).

Intuitively, observe that whenever a new honest block  $b$  is created,  $future_h(z, G_{time(b)}^{oracle})$  increases by 1 for all  $z$ 's in its past. For  $b$  itself, the value of this variable is 0. Thus, the value of  $\max_{z \in G_s^{oracle} \cap honest} \{future_a(z, G_s^{oracle}) - future_h(z, G_s^{oracle})\}$  is lower bounded by 0. On the other hand, whenever a new attack block is created, the value of  $future_a(z, G_s^{oracle})$  increases by 1 for all honest blocks available to it at the time (following the worst case assumptions specified in Lemma 12, *Part I*). Therefore, the attacker's maximal advance over the honest network can be modeled as a reflecting random walk. Note that, since the creation of an honest block  $b$  increases  $future_h(z, G_s^{oracle})$  only for blocks in  $past_h(b)$ , which might be a proper subset of  $before(time(b))$  (when  $d > 0$ ), there are certain situations where honest blocks do not "work against" attack blocks to decrease the value of  $\max \{|future_a(z, G_s^{oracle})| - |future_h(z, G_s^{oracle})|\}$ . We take this into account by skewing the behaviour of the walk whenever the origin is visited (and proving that in all other states the honest network's inner delay has no effect).

In the following analysis, we assume the worst case scenario, namely, that if  $z_1$  and  $z_2$  are honest blocks such that  $|time(z_1) - time(z_2)| < d$  then  $z_1 \in anticone(z_2)$ . That this is a worst case follows simply from the fact that omitting some edges between honest blocks can only decrease  $future_h(z, G_s^{oracle})$  hence increase the value of  $|future_a(z, G_s^{oracle})| - |future_h(z, G_s^{oracle})|$ .

If the attacker is creating blocks in secret, it needs to decide upon a strategy regarding which blocks should its new block point at, for every new block it creates. Consider the following strategy: The attacker's new block  $b$ , created at  $time(b)$ , points at  $G_{time(b)}^{oracle}$  (except itself, of course). While we have already argued why this is a worst case assumption (Lemma 12, *Part I*), it is here easy to see that this strategy maximizes  $\max_{z \in G_s^{oracle} \cap honest} \{|future_a(z, G_s^{oracle})| - |future_h(z, G_s^{oracle})|\}$ .

**Part II:** Denote by  $t_i$  the creation time of the  $i$ th block in  $G_r^{oracle}$ . Denote by  $z_s$  the variable  $\arg \max_{z \in G_s^{oracle} \cap honest} \{|future_a(z, G_s^{oracle})| - |future_h(z, G_s^{oracle})|\}$ . Define further  $A_s^z := |future_a(z, G_s^{oracle})|$ , and  $H_s^z := |future_h(z, G_s^{oracle})|$ . Abbreviate  $A_s := |future_a(z_s, G_s^{oracle})|$ , and  $H_s := |future_h(z_s, G_s^{oracle})|$ .

We define a subseries  $(s_k) \subseteq (t_i)$  recursively:  $s_0 = 0$ , and for all  $k > 0$ :  $s_{k+1} = \min_i \{t_i : t_i \geq time(z_{s_k}) + d\}$ . We claim that  $(A_{s_k} - H_{s_k})$  has the same probability distribution as  $X_k$ . Assume this claim holds true, and let  $s_k$  be the earliest  $s_k$  with

$s_k \geq r$ . Then  $(A_r - H_r) \leq 1 + (A_{s_k} - H_{s_k})$ .<sup>22</sup> Consequently,

$$\begin{aligned} & \Pr \left( \max_{z \in G_r^u \cap \text{honest}} \{|future_a(z, G_r^u)| - \right. \\ & \quad \left. |future_h(z, G_r^u)|\} > l \right) = \\ & \Pr(A_r - H_r > l) \leq \Pr(A_{s_k} - H_{s_k} > l - 1) = \\ & \Pr(X_k > l - 1) = 1 - \Pi((l - 1)^+). \end{aligned}$$

**Part III:** To complete the proof we prove our claim, by induction on  $k$ . For  $k = 0$ ,  $s_0 = 0$ . At time 0, following the creation of the *genesis* block, the value of  $(A_0 - H_0)$  is 0, as  $future(\text{genesis}) \cap G_0^{\text{oracle}} = \emptyset$ , and likewise  $X_0 = 0$ . Assume we have proved this for  $k$ , and we now prove it for  $k + 1$ . Assume first that  $(A_{s_k} - H_{s_k}) > 0$ . Assume by way of negation that  $s_k < \text{time}(z_{s_k}) + d$ . Then, by the construction of  $(s_k)$ ,  $s_k = \text{time}(z_{s_k})$ . This implies that the honest network created  $z_{s_k}$  in time  $s_k$ . Thus  $(A_{s_k} - H_{s_k}) = 0$ , because  $z_{s_k}$  was created at time  $s_k$ . As  $z_{s_k}$  is supposed to be in  $\arg \max_{z \in G_r^{\text{oracle}} \cap \text{honest}} \{A_{s_k} - H_{s_k}\}$ , this contradicts our assumption that  $A_{s_k} - H_{s_k} > 0$ . Thus,  $(A_{s_k}^z - H_{s_k}^z) > 0$  implies  $s_k \geq \text{time}(z_{s_k}) + d$ .

Consequently, if  $(A_{s_k} - H_{s_k}) > 0$ , we are guaranteed that the entire honest network has learnt about the block  $z_{s_k}$ . Thence, the honest network adds blocks to  $future(z_{s_k})$  at a rate of  $(1 - \alpha \cdot \lambda)$ , while the attacker adds them at a rate of  $\alpha$ . Every block of the honest network then contributes 1 to  $|future_h(z, G_s^{\text{oracle}})|$ , whereas an attacker block contributes 1 to  $|future_a(z, G_s^{\text{oracle}})|$ . Thus,  $(A_{s_k} - H_{s_k})$  increases by 1 by the addition of an attacker's block, that is, w.p. $\alpha$ , and decreases by 1 w.p. $(1 - \alpha)$ . Indeed, conditioned on  $X_k > 0$ ,  $X_{k+1}$ 's distribution behaves the same:  $\Pr(X_{k+1} = X_k + 1 \mid X_k > 0) = 1 - \Pr(X_{k+1} = X_k - 1 \mid X_k > 0) = \alpha$ .

Assume now that  $(A_{s_k} - H_{s_k}) = 0$ . It cannot be the case that the block that was created in time  $s_k$  belongs to the attacker, since that would imply that the attacker has an advantage of at least 1 over the last block that was created by the honest network (up to time  $s_k$ ). Therefore, it belongs to the honest network. By the definition of  $z_{s_k}$ , it is precisely the block that was created in time  $s_k$ . Consequently, in the interval  $(s_k, s_k + d)$ , the honest network does not add blocks to  $future(z_{s_k})$  (recall we are assuming that the worst case scenario is realized, i.e., a propagation time of  $d$  seconds per honest block). During this interval, the attacker creates blocks at following a Poisson process with parameter  $\alpha \cdot \lambda$ . Thus,  $(A_{s_k+d}^{z_{s_k}} - H_{s_k+d}^{z_{s_k}}) = i$  w.p.  $\mathcal{P}_{\text{Pois}}(\alpha \cdot \lambda \cdot d, i)$ . Upon which, the next block in the system, created after  $s_k + d$ , is the attacker's w.p. $\alpha$ , in case which the total gap increases by  $j + 1$ , i.e.,  $(A_{s_{k+1}}^{z_{s_{k+1}}} - H_{s_{k+1}}^{z_{s_{k+1}}}) = (A_{s_k+d}^{z_{s_k}} - H_{s_k+d}^{z_{s_k}}) + 1$ ; alternatively, the next block after  $s_k + d$  is the honest network's, w.p. $(1 - \alpha)$ , in case which  $(A_{s_{k+1}}^{z_{s_{k+1}}} - H_{s_{k+1}}^{z_{s_{k+1}}}) = \max \left\{ (A_{s_k+d}^{z_{s_k}} - H_{s_k+d}^{z_{s_k}}) - 1, 0 \right\}$ . By comparing this to  $\Pr(X_{k+1} \mid X_k = 0)$ , we see that also in this case the variable  $X_{k+1}$

<sup>22</sup>Indeed, if  $r = s_k$  then this holds trivially. Otherwise, in the interval  $(r, s_k)$  the honest network could have contributed at most one block to  $future(z_{s_{k-1}})$ , because  $(X_i)$  can decrease by at most 1 at every step, according to its transition matrix, thus in the interval  $(r, s_k) \subseteq (s_{k-1}, s_k)$  the honest network created at most 1 block.

behaves the same as  $\left(A_{s_{k+1}}^{z_{s_{k+1}}} - H_{s_{k+1}}^{z_{s_{k+1}}}\right)$ .  $\square$

**Lemma 24.**  $f_{pre\_mine}(l) \leq C_l \cdot e^{-B_l \cdot future_a(x, G_l^v)}$  for some positive constants  $B_l, C_l$ .

To get the intuition of this result, notice that when  $d = 0$  the stationary distribution of the reflecting random walk is known to be proportionate to  $\left(\frac{\alpha}{1-\alpha}\right)^l$ , and if  $d > 0$ , this relation still holds for  $l \gg d \cdot \lambda$ .

*Proof.* For  $n > 1$ , the stationary distribution  $\pi$  satisfies the relation  $\pi(n) = (1 - \alpha) \cdot \pi(n + 1) + \alpha \cdot \pi(n - 1) + e^{-\delta} \cdot \frac{\delta^n}{n!} \cdot \pi(0)$ . Let us write  $\pi(n) = C_n \cdot \left(\frac{\alpha}{1-\alpha}\right)^n$  for  $n \geq 0$ . We have:

$$C_n \cdot \left(\frac{\alpha}{1-\alpha}\right)^n = (1 - \alpha) \cdot C_{n+1} \cdot \left(\frac{\alpha}{1-\alpha}\right)^{n+1} + \quad (37)$$

$$\alpha \cdot C_{n-1} \cdot \left(\frac{\alpha}{1-\alpha}\right)^{n-1} + e^{-\delta} \cdot \frac{\delta^n}{n!} \cdot \pi(0) \implies \quad (38)$$

$$C_n = C_{n+1} \cdot \alpha + C_{n-1} \cdot (1 - \alpha)^{-1} + e^{-\delta} \cdot \frac{\left(\delta \cdot \frac{(1-\alpha)}{\alpha}\right)^n}{n!} \cdot \pi(0). \quad (39)$$

For large enough  $n$ 's, the last summand in the above expression is negligible. Thus, when we write  $\forall n : C_n \approx C$ , the above relation will be satisfied for large  $n$ 's (up to the negligible error of the last summand). Thus, for some constant  $C$ ,  $\pi(n) \leq C \cdot \left(\frac{\alpha}{1-\alpha}\right)^n$ , hence  $1 - \Pi((n-1)^+) = \sum_{k=n}^{\infty} \pi(k) \leq B_l \cdot e^{-C_l \cdot n}$ , for some large enough  $n$ , and some constants  $B_l, C_l > 0$ .  $\square$

The following Corollary is immediate from Lemma 23.

**Corollary 25.** *In Lemma 20, if  $l$  is not known, then*

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathbb{G}}\right) \leq \sum_{l=0}^{\infty} \pi(l) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot \lambda, h). \quad (40)$$

$$\sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{(g-h-l-m)^+}. \quad (41)$$

Similarly, in Corollary 21,

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathbb{G}}\right) \leq \sum_{l=0}^{\infty} \pi(l) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot \alpha \lambda, h). \quad (42)$$

$$\sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m. \quad (43)$$

$$\left(\frac{\alpha}{1 - \alpha}\right)^{(g-h-l-m)^+}. \quad (44)$$



Finally, in Corollary 22,

$$\Pr\left(\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathcal{G}}\right) \leq \sum_{l=0}^{\infty} \pi(l) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}((3 - 2 \cdot \alpha) \cdot d \cdot \lambda, h) \cdot \sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{(n_x - h - l - m)^+}.$$

Using this corollary we can prove the bound that is used by the online policy described in Algorithm 7. Denote:

$$risk\_hidden(T, g) := \sum_{l=0}^{\infty} \pi(l) \cdot \sum_{m=0}^{\infty} \mathcal{P}_{oiss}((T + 2 \cdot d) \cdot \alpha \cdot \lambda). \quad (45)$$

$$\left(\frac{\alpha}{1 - \alpha}\right)^{(g - l - m)^+} \quad (46)$$

**Corollary 26.** *If Algorithm 7 returns a value less than  $\epsilon$  then  $\Pr\left(\bigcup_{y \in G_{\infty}^{pub} \setminus G_t^{pub}} \widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathcal{G}}\right) < \epsilon$ .*

*Proof.* First, observe that the variable  $g$  used in Corollary 21 could be replaced (here and in that corollary) with  $\min_{x' \in \overline{anticoner}_n(x, G_t^v)} |future(x', G_t^v)|$ , because all blocks in  $future(x, G_t^v)$  vote in favour of  $x$ , by the assumption on  $y$ . The value assigned to  $g$ , in line 5 of Algorithm 7, is upper bounded by  $\min_{x' \in \overline{anticoner}(x, G_x)} |future(x', G_x)|$ , because  $G_x$  includes all honest blocks in  $G_t^v$ . Next,  $T$  is assigned the value  $time\_now - received^v(x) = t - received^v(x)$ , in line 3. Observe that  $m$  is distributed according to  $\mathcal{P}_{oiss}(m, (t - time(x)) \cdot \alpha \cdot \lambda)$ .<sup>23</sup> As  $time(x) \geq received^v(x) + d$ , we can upper bound this by a Poisson variable with parameter  $(T + d) \cdot \alpha \cdot \lambda$ . We then adjust the result of the second term in Corollary 25; we combine the distributions over  $h$  and  $m$  (where  $h$  is taken from (42), to conclude that

$$\Pr\left(\bigcup_{y \in G_{\infty}^{pub} \setminus G_t^{pub}} \widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathcal{G}}\right) \leq \quad (47)$$

$$\sum_{l=0}^{\infty} \pi(l) \cdot \sum_{m=0}^{\infty} \mathcal{P}_{oiss}((T + 2 \cdot d) \cdot \alpha \cdot \lambda) \cdot \left(\frac{\alpha}{1 - \alpha}\right)^{(g - l - m)^+} = \quad (48)$$

$$risk\_hidden(T, g). \quad (49)$$

Note that we do not need to apply here a union bound over the different  $y$ 's in  $G_{\infty}^{pub} \setminus G_t^{pub}$ , because our analysis assumes that in the worst case all of the attacker blocks vote strongly in favour of  $y \prec x$ , for all  $y$  in this set, and, additionally, all honest blocks in  $G_t^v$  will always vote strongly in favour of  $x \prec y$ , for all  $y$ 's in this set (as they do not see  $y$  in their past). Thus, under our worst case analysis, the event where for some  $y$  in  $G_{\infty}^{pub} \setminus G_t^{pub}$  the attacker manages to reverse the relation  $x \prec y$  is equivalent to the event where it manages to do so for a given  $y$ . In conclusion, if Algorithm 7 returned a value less than  $\epsilon$ , we know that  $\Pr\left(\bigcup_{y \in G_{\infty}^{pub} \setminus G_t^{pub}} \widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)^{\mathcal{G}}\right) < \epsilon$ .  $\square$

<sup>23</sup>Our previous analysis measured  $m$  using  $n_x$ , as it was structure-based and had no access to  $T$ .

Denote by  $dist\_gap(b, G)$  the minimal  $k$  for which  $gap(b, \langle G, b, K \rangle) = 0$ .

**Lemma 27.** *Let  $b$  be an honest block. Then,*

$$\Pr \left( \bigcup_{u \in honest, s \in [time(b), \infty)} dist\_gap(b, G_s^u) > K \right) \leq \quad (50)$$

$$\begin{aligned} & \sum_{l=0}^{\infty} \pi(l) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot \lambda, h) \cdot \\ & \sum_{m=0}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m \cdot \\ & \left( \frac{\alpha}{1 - \alpha} \right)^{(K - h - l - m)^+}. \end{aligned} \quad (51)$$

We denote the RHS of (50) by  $f_{distgap}(K)$ .

*Proof.* By its definition, the event where  $dist\_gap(b, G_s^u) > K$  is equivalent to the event where some block in  $anticone\left(b, G_{time(b)}^{node(b)}\right) \cup \left(G_s^u \setminus G_{time(b)}^{node(b)}\right)$  precedes  $b$  (or achieves a tie with it) according to  $vote(virtual(\langle G_s^u, b, K \rangle))$ . In  $\langle G_s^u, b, K \rangle$ ,  $b$  has additional  $K$  blocks  $b_1, \dots, b_K$  that vote in favour of it against any other  $y \notin \overline{past}(b)$ ; indeed, for any  $y \notin \overline{past}(b)$ ,  $y \notin \overline{past}(b_i)$ . Consequently, at  $G_{time(b)}^{node(b)}$ ,  $K$  blocks in  $future(b)$  vote in its favour against any block in its anticone. We can thus apply the first part of Corollary 25 with  $n_x = K$ ,  $X = \{b\}$ , and  $Y = G_{\infty}^{oracle} \setminus \overline{past}(b)$  to conclude that (50) is an upper bound on the probability that a block in  $anticone\left(b, G_{time(b)}^{node(b)}\right)$  will ever precede  $b$  (or obtain a tie with it) in the pairwise order of  $\langle G_s^u, b, k \rangle$  for any  $s \geq time(b)$  in the future.  $\square$

**Lemma 28.** *For all  $n_x \in \mathbb{N}$ ,*

$$\Pr(|future_h(x, G_t^v)| > n_x) \leq \quad (52)$$

$$|future(x, G_t^v)| \cdot f_{distgap}\left(\sqrt{|future(x, G_t^v)|}\right). \quad (53)$$

The RHS of the last inequality is denoted  $f_{post\_pub}(|future(x, G_t^v)|)$ .

*Proof.* If  $y = NULL$  there is nothing to prove, since then  $n_x = future(x, G_t^v) \geq future_h(x, G_t^v)$ . Assume  $y \neq NULL$ .

Denote  $K := \sqrt{|future(x, G_t^v)|}$ .  $n_x$  is obtained in Algorithm 3 by subtracting  $M$ , the number of blocks with  $dist\_gap > K$ , from  $future(x, G_t^v)$ . Let  $b$  be an honest block in  $future(x, G)$ . By Lemma 27, the probability that  $dist\_gap(b, G_t^v)$  will be larger than  $K$  is at most  $f_{distgap}(K)$ . By the union bound, the probability that for some  $b$  in  $future_h(x, G_t^v)$ ,  $dist\_gap(b, G_t^v) > K$ , is at most  $|future_h(x, G_t^v)| \cdot f_{distgap}(K) \leq |future(x, G_t^v)| \cdot f_{distgap}(K) = f_{post\_pub}(|future(x, G_t^v)|)$ .  $\square$

The RHS of Inequality (50) implies:

**Lemma 29.**  $f_{post\_pub}(|future(x, G_t^v)|) \leq C_c \cdot e^{-B_c \cdot future_a(x, G_t^v)}$  for some positive constants  $B_c, C_c$ .

**Lemma 30.** For  $n_j \in \mathbb{N}$  put  $j := \text{gap}(x, G) + n_j$ .

$$\Pr(\{| \text{anticone}_h(x, G_t^{\text{oracle}}) | > j\}) \leq \quad (54)$$

$$f_{\text{pre\_mine}}(\sqrt{n_j}) + \sum_{h'=0}^{\infty} \mathcal{P}_{\text{oiiss}}((1-\alpha) \cdot \lambda \cdot d, h'). \quad (55)$$

$$f_{\text{post\_mine}}(n_j, n_j - h' + 1, \sqrt{n_j}) \quad (56)$$

We denote the RHS of this inequality by  $f_{\text{pre\_pub}}(n_j)$ . To understand the intuition behind this result recall that w.h.p. a block defeats only blocks that were published close to its publication or after it.

*Proof. Part I:* Let  $t_x := \text{publication}(x)$ . Define  $L_n := \{z \in \text{anticone}_h(x, G_t^v) : \text{future}_h(z, \text{anticone}_h(x, G_{t_x}^v)) \geq n\}$ . (Note the use of  $t_x$  in this definition). Denote by  $A_{n'}$  the event  $\{\exists z \in L_n : z \in X_{\text{win}}(x, G_t^v)\}$ . Finally, let  $z_e$  be the earliest block in  $L_{n'}^{\mathbb{C}} \cap \text{anticone}_h(x, G_t^{\text{oracle}})$  and put  $n' := n_j - |\overline{\text{anticone}_h(z_e, G_t^{\text{oracle}})}| + 1$  for  $n_j := \sqrt{|\text{future}_h(x, G_t^v)|}$ . Denote by  $X_{\text{win}}(x, G)$  the set of blocks that  $x$  precedes (or obtains a tie with) in the pairwise order of  $G$ 's virtual vote, and by  $X_{\text{lose}}(x, G)$  the rest of blocks. Then:

$$\begin{aligned} & \{|\text{anticone}_h(x, G_t^{\text{oracle}})| > \text{gap}(x, G) + n_j\} = \\ & \left\{ \left| X_{\text{win}}(x, G_t^{\text{pub}}) \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \right| + \right. \\ & \left. \left| X_{\text{lose}}(x, G_t^{\text{pub}}) \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \right| > \text{gap}(x, G) + n_j \right\} = \\ & \left\{ \left| X_{\text{win}}(x, G_t^{\text{pub}}) \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \right| + \text{gap}(x, G_t^v) > \text{gap}(x, G) + n_j \right\} = \\ & \left\{ \left| X_{\text{win}}(x, G_t^{\text{pub}}) \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \right| > n_j \right\} = \\ & \left( \left\{ \left| X_{\text{win}}(x, G_t^{\text{pub}}) \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \right| > n_j \right\} \cap A_{n'} \right) \cup \\ & \left( \left\{ \left| X_{\text{win}}(x, G_t^{\text{pub}}) \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \right| > n_j \right\} \cap A_{n'}^{\mathbb{C}} \right) \subseteq \\ & A_{n'} \cup \left\{ \left| L_{n'}^{\mathbb{C}} \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \right| > n_j \right\} = \\ & A_{n'} \cup \left\{ \left| L_{n'}^{\mathbb{C}} \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \cap \overline{\text{anticone}_h(z_e, G_t^{\text{oracle}})} \right| + \right. \\ & \left. \left| L_{n'}^{\mathbb{C}} \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \cap \text{future}_h(z_e, G_t^{\text{oracle}}) \right| > n_j \right\} = \\ & A_{n'} \cup \left\{ \left| L_{n'}^{\mathbb{C}} \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \cap \text{future}_h(z_e, G_t^{\text{oracle}}) \right| > \right. \\ & \left. n_j - \left| L_{n'}^{\mathbb{C}} \cap \text{anticone}_h(x, G_t^{\text{oracle}}) \cap \overline{\text{anticone}_h(z_e, G_t^{\text{oracle}})} \right| \right\} \subseteq \\ & A_{n'} \cup \left\{ \left| \text{anticone}_h(x, G_t^{\text{oracle}}) \cap \text{future}_h(z_e, G_t^{\text{oracle}}) \right| > \right. \\ & \left. n_j - \left| \overline{\text{anticone}_h(z_e, G_t^{\text{oracle}})} \right| \right\}. \end{aligned}$$

As  $z_e \in L_{n'}$ , and by the definition of  $n'$ , it cannot be the case that  $\text{anticone}_h(x, G_t^{\text{oracle}}) \cap \text{future}_h(z_e, G_t^{\text{oracle}})$  contains

more than  $n_j - |\overline{anticoner_h}(z_e, G_t^{oracle})|$  blocks. Thus, the event  $\{|\overline{anticoner_h}(x, G_t^{oracle}) \cap future_h(z_e, G_t^{oracle})| > n_j - |\overline{anticoner_h}(z_e, G_t^{oracle})|\}$  occurs w.p. 0, and we obtain:  $\Pr(\{|anticoner_h(x, G_t^{oracle})| > gap(x, G) + n_j\}) \leq \Pr(A_{n'})$ .

Observe that all blocks in  $future(z, anticoner_h(x, G_{t_x}^v))$  vote strongly in favour of  $z$  against  $x$ , for any  $z \in L_{n'}$ , and that by definition there are at least  $n'$  such votes at time  $t_x$ . Consequently, we can apply the result of Corollary 22 with respect to the following parameters:  $v = pub$ ,  $t = t_x$ ,  $X =$  the leaf-blocks of  $L_{n'}$ ,  $Y = \{x\}$ ,  $g := n'$ ,  $n_x := n_j$ , and  $l' = \max_{z \in G_{t_x}^{oracle} \cap honest} \{|future_a(z, G_{t_x}^{oracle})| - |future_h(z, G_{t_x}^{oracle})|\}$ , to obtain:

$$\begin{aligned} \Pr(A_{n'}) &= \Pr(\exists z \in L_{n'} : z \in X_{win}(x, G_t^v)) \leq \\ &\Pr(\exists s > t_x, \exists z \in L_{n'} : z \in X_{win}(x, G_s^v)) \leq \\ &f_{post\_mine}(n_j, n', l'). \end{aligned}$$

As the value of  $l'$  is unknown to us, we use Lemma 23 to conclude that with probability  $\geq 1 - f_{pre\_mine}(l)$  its value is at most  $l$ . Fix  $l = \sqrt{n_j}$ . Similarly, the value of  $n'$  is unknown to us. However, blocks in  $\overline{anticoner_h}(z_e, G_t^{oracle})$  are created in the time interval  $[time(z_e), time(z_e) + d]$  (by its choice), hence  $|\overline{anticoner_h}(z_e, G_t^{oracle})|$  is a Poisson variable with parameter  $(1 - \alpha) \cdot \lambda \cdot d$ . We thus conclude that:

$$\begin{aligned} \Pr(\{|anticoner_h(x, G_t^{oracle})| > gap(x, G) + n_j\}) &\leq \\ f_{pre\_mine}(\sqrt{n_j}) + \sum_{h'=0}^{\infty} \mathcal{P}_{oiss}((1 - \alpha) \cdot \lambda \cdot d, h') \cdot \\ f_{post\_mine}(n_j, n_j - h' + 1, \sqrt{n_j}) &= \\ f_{pre\_pub}(n_j). \end{aligned}$$

□

It is easy to verify that  $f_{post\_mine}(n_j, n_j - h' + 1, \sqrt{n_j})$  decreases exponentially (we do this in fact in subsequent lemmas). Therefore:

**Lemma 31.**  $f_{pre\_pub}(n_j) \leq C_j \cdot e^{-B_j \cdot n_j}$  for some positive constants  $B_j, C_j$ .

In the lemma below,  $oracle^u$  is a (hypothetical) node such that  $G_s^{oracle^j} := G_s^u \cup (G_s^{oracle} \cap attacker)$ .

**Lemma 16.** *Conditioned on the event  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$ , there exists a time  $\tau \in [t, \infty)$  such that  $\forall s \geq \tau: M(oracle^u, s) \geq |future_a(x, G_s^{oracle^u}) \cap G_{[t, s]}^{oracle} \setminus V_{x \prec y}(G_s^{oracle^u})| - m^*$ , for some  $m^*$  that remains fixed after  $\tau$  (and with  $\mathbb{E}[m^*]$  determined by the events up to time  $t$ ).*

*Proof. Part I:* If  $y \notin G_s^{oracle^u}$  then  $M(oracle^u, s) = 0$  (line 6),  $V_{x \prec y}(G_s^{oracle^u}) = future(x, G_s^{oracle^u})$ , and the required inequality follows trivially. Assume  $y \in G_s^{oracle^u}$ .

Let  $G$  be any block DAG that equals the past-set of some (possibly virtual) block. Observe that conditioned on  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$ , for some constant  $C_t$  determined at time  $t$ , if  $|G_{[t,s]} \cap \text{attacker}| - |G_{[t,s]}^{oracle^u} \cap \text{honest}| < -C_t$  then  $\text{vote}_{x,y}(\text{virtual}(G)) = -1$ .<sup>24</sup> This follows from the proof of Lemma 12: We take the LHS of (17), replace  $g$  by  $\sum_{z' \in \text{future}(x, G_t)} \text{vote}_{x,y}(z', G)$ , and observe that the value of the remaining term  $2 \cdot \left| \text{anticon}_h(z_{late}, G_{time(z_{late})}^{oracle^u}) \right| - \left| \text{future}_h(z, G_{time(z_{late})}^{oracle^u}) \right| + \left| \text{future}_a(z, G_{time(z_{late})}^{oracle^u}) \right| + \left| \text{anticon}_h(z_{late}, G_{[time(z_{late}), time(z_{late})+d]}^{oracle^u}) \right| + \left| G_{[t-d,t]}^{oracle} \cap \text{honest} \right| + \left| \text{future}_h(z_{late}, G_t^v) \setminus \text{future}_h(x, G_t^v) \right| + \left| \text{future}_a(z_{late}, G_t^{oracle^u}) \setminus \text{future}_a(x, G_t^v) \right|$  is determined by time  $t$ , hence we can denote it  $C_t$ .

Let  $z \in \text{future}(x, G_{t,s}^{oracle})$ . By the conditioning on  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$ ,  $z \in \text{attacker}$ . Fix the DAG  $G^z := \text{past}(z)$ . The above argument holds in particular for  $G^z$ : If  $|G_{[t,s]}^z \cap \text{attacker}| - |G_{[t,s]}^z \cap \text{honest}| < -C_t$  then  $\text{vote}_{x,y}(z) = \text{vote}_{x,y}(\text{virtual}(\text{past}(z))) = -1$  (since  $z$  is a strong voter we do not need to specify the context of its vote). Consequently, if  $z \in G_{t,s}^{oracle} \setminus V_{x \prec y}(G_s^{oracle})$  then  $\left| G_{[t,time(z)]}^z \cap \text{attacker} \right| - \left| G_{[t,time(z)]}^z \cap \text{honest} \right| \geq -C_t$ .<sup>25</sup>

We arrive at the following important implication: If  $z \in \text{future}(x, G_s^{oracle^u}) \setminus V_{x \prec y}(G_s^{oracle^u})$  then:

$$\left| \text{anticon}_e(z, G_s^{pub}) \right| \geq \left| \text{anticon}_e(z, G_{time(z)}^{pub}) \right| \geq \quad (57)$$

$$\left| \text{anticon}_e(z, G_{time(z)}^{pub}) \setminus G_t^{oracle} \right| = \quad (58)$$

$$\left| G_{time(z)}^{pub} \setminus G_t^{oracle} \right| - \left| \overline{\text{past}}(z) \setminus G_t^{oracle} \right| \geq \quad (59)$$

$$\left| G_{time(z)}^{pub} \cap \text{honest} \setminus G_t^{oracle} \right| - \left| G_{[t,time(z)]}^z \cap \text{honest} \right| \geq \quad (60)$$

$$\left| G_{[t,time(z)]}^{pub} \cap \text{honest} \right| - \left| G_{[t,time(z)]}^z \cap \text{attacker} \right| - C_t. \quad (61)$$

**Part II:** Let  $z_1, z_2, \dots$  the order of creation of blocks in  $\text{future}_a(x, G_s^{oracle^u} \setminus G_t^{oracle}) \setminus V_{x \prec y}(G_s^{oracle^u})$ . Fix  $z_m$ , and let  $b_m$  be the earliest block in  $\text{anticon}_h(z_m, \text{future}(x, G_s^{oracle^u}))$ . With probability  $\mathcal{P}_{\text{oiSS}}(d \cdot (1-\alpha) \cdot \lambda, h')$ ,

<sup>24</sup>We write here  $G_{[t,s]}^z$  for  $G^z \cap \text{before}(s) \setminus \text{before}(t)$ .

<sup>25</sup>Note that  $G^z$  contains only blocks created up to  $\text{time}(z)$ .

$|anticone_h(b_m, G_\infty^{pub})| = h'$ . By the choice of  $b_m$  together with (57) we obtain:

$$\begin{aligned}
& \left| future_h(b_m, G_{time(z_m)}^{pub}) \right| = \\
& \left| anticone_h(z_m, G_{time(z_m)}^{pub}) \setminus anticone_h(b_m, G_{time(z_m)}^{pub}) \right| \geq \\
& \left| anticone_h(z_m, G_{time(z_m)}^{pub}) \right| - \left| anticone_h(b_m, G_{time(z_m)}^{pub}) \right| \geq \\
& \left| G_{[t, time(z_m)]}^{pub} \cap honest \right| - \left| G_{[t, time(z_m)]}^{z_m} \cap attacker \right| - C_t - h' = \\
& \left| G_{[t, time(z_m)]}^{pub} \cap honest \right| - m - C_t - h', \tag{62}
\end{aligned}$$

where we used the fact that  $past(b_m) \cap anticone_h(z_m) = \emptyset$ , by the choice of  $b_m$ , and that  $anticone_h(z_m, G_{time(z_m)}^{pub}) = antipast_h(z_m, G_{time(z_m)}^{pub})$ .

**Part III:** Given  $m$ ,  $\left| G_{[t, time(z_m)]}^{pub} \cap honest \right|$  is distributed according to a negative binomial distribution:  $\Pr\left(\left| G_{[t, time(z_m)]}^{pub} \cap honest \right| = n\right) = \binom{n+m-1}{n} \cdot (1-\alpha)^n \cdot \alpha^m$ . We claim that the probability that the honest block  $b_m$  will ever be preceded by  $z_m$  in the order of *virtual* ( $\langle G_s^{oracle^u}, z_m, K \rangle$ ) is at most

$$\begin{aligned}
& \sum_{l=0}^{\infty} \pi(l) \cdot \sum_{k=0}^{\infty} \mathcal{P}_{oiss}(5 \cdot d \cdot (1-\alpha) \cdot \lambda, k) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot (1-\alpha) \cdot \lambda, h) \cdot \\
& \sum_{n=0}^{\infty} \binom{n+m-1}{m} \cdot (1-\alpha)^n \cdot \alpha^m \cdot \left(\frac{\alpha}{1-\alpha}\right)^{(n-m-K-2 \cdot h-C_t-k-l)^+}. \tag{63}
\end{aligned}$$

This follows from a similar analysis to that made in the proof of Lemma 12 and of Corollary ???. Indeed, at  $time(z_m)$  there were at least  $n - m - h' - C_t$  blocks in  $future(b_m) \setminus \overline{future}(z_m)$ , by the above lower bound on  $\left| future_h(b_m, G_{time(z_m)}^{pub}) \right|$ ; and while  $future_a(b_m, G_{time(z_m)}^{pub}) = \emptyset$ , as  $b_m \notin past(z_m)$ , there are additional  $K$  hypothetical blocks that vote  $y \prec x$ , by the construction of  $\langle G_s^{oracle^u}, z_m, K \rangle$ . Instead of reducing  $h'$  in the exponent (as in the bound given in Corollary ??), we added  $2 \cdot d \cdot (1-\alpha) \cdot \lambda$  to the variable  $k$ , as the sum of Poisson variables is a Poisson variable. Finally, we use the result of Lemma 23 to ensure that  $\pi(l)$  upper bounds the distribution over  $l$ ,<sup>26</sup>

As  $dist\_gap(G_s^{oracle^u}, z_m) \leq K$  requires  $z_m$  to precede  $b_m$  in the order of *virtual* ( $\langle G_s^{oracle^u}, z_m, K \rangle$ ), (63) serves as an upper bound also to the probability that  $dist\_gap(G_s^{oracle^u}, z_m) \leq K$ .

**Part IV:** Using Lemma 24 it is easy to verify the existence of constants  $a, b$ , and  $W$  such that  $\Pr(k + l + 2 \cdot h > W) \leq e^{-a \cdot W + b}$ .

Put  $K(oracle^u, s) = \sqrt{|future(x, G_s^{oracle^u})|}$ . The block  $z_m$  is counted into  $M(oracle^u, s)$  in line 9 of Algorithm 3 whenever  $dist\_gap(G_s^{oracle^u}, z_m) >$

<sup>26</sup> $l$  here represents  $\max_{z \in G_{time(b_m)}^{oracle} \cap honest} \left\{ \left| future_a(z, G_{time(b_m)}^{oracle^u}) \right| - \left| future_h(z, G_{time(b_m)}^{oracle^u}) \right| \right\}$ .

$K(\text{oracle}^u, s)$ . From (63) we conclude that the probability that  $z_m$  does *not* increment by 1 the value of  $M(\text{oracle}^u, s)$  is upper bounded by

$$\Pr\left(\text{dist\_gap}(G_s^{\text{oracle}^u}, z_m) \leq K(\text{oracle}^u, s)\right) \leq \quad (64)$$

$$\sum_{n=0}^{\infty} \binom{n+m-1}{m} \cdot (1-\alpha)^n \cdot \alpha^m \cdot \left(\frac{\alpha}{1-\alpha}\right)^{(n-m-K(\text{oracle}^u, s)-W-C_t)^+} < \\ \left(\frac{\alpha}{1-\alpha}\right)^{-W-C_t-K(\text{oracle}^u, s)} \cdot \sum_{n=0}^{\infty} \binom{n+m-1}{n} \cdot (1-\alpha)^n \cdot \alpha^m \cdot \left(\frac{\alpha}{1-\alpha}\right)^{(n-m)^+} = \quad (65)$$

$$\left(\frac{\alpha}{1-\alpha}\right)^{-W-C_t-K(\text{oracle}^u, s)} \cdot \left(\Pr_{n \sim Z(m, 1-\alpha)}(n > m) + \Pr_{n \sim Z(m, \alpha)}(n \leq m)\right), \quad (66)$$

where  $Z(n, p)$  denotes a negative binomial random variable.

We now aim at showing that the last term is upper bounded by some  $e^{-D \cdot m}$ . The proof is very similar to that given in Lemma 17 below.

**Part V:** For large enough  $m$ 's, a variable distributed according to  $Z(1-\alpha, m)$  converges to a normal variable with mean  $m \cdot \frac{\alpha}{1-\alpha}$  and variance  $m \cdot \frac{\alpha}{(1-\alpha)^2}$ .<sup>27</sup> The second multiplicand in (66) thus converges, as  $m$  grows, to

$$\Pr_{z \sim \mathcal{N}(0,1)}\left(z \leq \frac{m - \frac{1-\alpha}{\alpha} \cdot m}{\sqrt{\frac{1-\alpha}{\alpha^2} \cdot m}}\right) + \Pr_{z \sim \mathcal{N}(0,1)}\left(z \geq \frac{m - \frac{\alpha}{1-\alpha} \cdot m}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot m}}\right) = \quad (67)$$

$$\Pr_{z \sim \mathcal{N}(0,1)}\left(z \geq \frac{\frac{1-\alpha}{\alpha} \cdot m - m}{\sqrt{\frac{1-\alpha}{\alpha^2} \cdot m}}\right) + \Pr_{z \sim \mathcal{N}(0,1)}\left(z \geq \frac{m - \frac{\alpha}{1-\alpha} \cdot m}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot m}}\right). \quad (68)$$

The following inequality is due to Komatu (1955). Let  $x \geq 0$  and let  $z \sim \mathcal{N}(0, 1)$ . Then:  $\Pr(z > x) \leq \frac{1}{\sqrt{2 \cdot \pi}} \cdot \frac{2 \cdot e^{-x^2/2}}{x + \sqrt{2+x^2}}$ . Put  $x_1 := \frac{1-2 \cdot \alpha}{\alpha} \cdot m$  and  $x_2 := \frac{1-2 \cdot \alpha}{\alpha} \cdot m$ . We obtain an upper bound on (68):

$$\frac{1}{\sqrt{2 \cdot \pi}} \cdot \frac{2 \cdot e^{-x_1^2/2}}{x_1 + \sqrt{2+x_1^2}} + \frac{1}{\sqrt{2 \cdot \pi}} \cdot \frac{2 \cdot e^{-x_2^2/2}}{x_2 + \sqrt{2+x_2^2}} \leq \quad (69)$$

$$C_1 \cdot e^{-x_1^2/2} + C_2 \cdot e^{-x_2^2/2} = C_1 \cdot e^{-D_1 \cdot m} + C_2 \cdot e^{-D_2 \cdot m} \leq C_3 \cdot e^{-D_3 \cdot m} \quad (70)$$

for some positive constants  $C_i, D_i$  that depend on  $\alpha$  (a property which applies to the constants below as well).

<sup>27</sup>We rely here on the assumption specified in the proof of Lemma 12 according to which, in the worst case, after time  $t$  the attacker publishes all his blocks to all nodes immediately after their creation.

When this term is multiplied by  $\left(\frac{\alpha}{1-\alpha}\right)^{-W-C_t-K(\text{oracle}^u, s)}$  we obtain

$$\left(\frac{\alpha}{1-\alpha}\right)^{-W-C_t-K(\text{oracle}^u, s)} \cdot C_3 \cdot e^{-D_3 \cdot m} \leq \quad (71)$$

$$C_4 \cdot e^{-D_3 \cdot m + D_4 \cdot K(\text{oracle}^u, s)} = C_4 \cdot e^{-D_3 \cdot m + D_4 \cdot \sqrt{|future(x, G_s^{\text{oracle}^u})|}}. \quad (72)$$

There exists therefore an  $M_1$  such that if  $m > |future(x, G_s^{\text{oracle}^u})| > M_1$  then the last expression is upper bounded by  $C_5 \cdot e^{-D_5 \cdot m}$  for some  $C_5, D_5$ .

**Part VI:** After some  $\psi$  (with expected value  $M_1/\lambda$ ), the condition  $|future(x, G_s^{\text{oracle}^u})| \geq M_1$  is satisfied. Put  $s_m := \text{time}(z_m)$  and assume  $s_m \geq \psi$ .

As  $\sum_{m=\sqrt{|future(x, G_s^{\text{oracle}^u})|}+1}^{\infty} C_5 \cdot e^{-D_5 \cdot m} < \infty$ , Fatou's lemma implies that

there exists (a.s.) an  $m^* > \sqrt{|future(x, G_{s_m}^{\text{oracle}^u})|}$  such that for all  $m \geq m^*$ ,  $\text{dist\_gap}(z_m) > K(\text{oracle}^u, s_m)$ . The expected waiting time for  $z_{m^*}$  is finite.<sup>28</sup> Define  $\tau = \max\{\psi, \text{time}(z_{m^*})\}$ . Then, for any  $s \geq \tau$ :  $M(\text{oracle}^u, s) \geq |future_a(x, G_s^{\text{oracle}^u}) \cap G_{[t, s]}^{\text{oracle}^u} \setminus V_{x \prec y}(G_s^{\text{oracle}^u})| - m^*$ .<sup>29</sup>  $\square$

**Lemma 17.** *There exists a  $\psi \in [t, \infty)$  such that  $\Pr(\mathcal{E}_{t \rightarrow \infty}^{\text{all}}(x, y, \epsilon)^{\complement} \mid \mathcal{E}_t^v(x, y, \epsilon)) < \epsilon$ . Moreover,  $\mathbb{E}[\psi] < \epsilon$ .*

*Proof. Part I:* We show that if all honest blocks vote in favour of  $x$  then all error functions converge to zero. Indeed, the event  $\mathcal{E}_t^v(x, y, \epsilon)$  implies that  $f_{\text{pre\_mine}}(l(G_t^v)) + f_{\text{pre\_pub}}(n_j(G_t^v)) + f_{\text{post\_pub}}(|future(x, G_t^v)|) + f_{\text{post\_mine}}(n_x(G_t^v), g(G_t^v), l(G_t^v)) < \epsilon$ . By the union bound, and by Lemmas 23, 28, and 30 respectively, the following relations hold with probability  $\geq 1 - \epsilon$ :

$$\bullet \max_{z \in G_t^{\text{oracle}^u} \cap \text{honest}} \left\{ \left| future_a(z, G_{\text{time}(x)}^{\text{oracle}^u}) \right| - \left| future_h(z, G_{\text{time}(x)}^{\text{oracle}^u}) \right| \right\} \leq l(G_t^v)$$

<sup>28</sup>We have  $\Pr(m^* \geq r) \leq \sum_{m=\sqrt{|future(x, G_s^{\text{oracle}^u})|}+1}^{r-1} C_5 \cdot e^{-D_5 \cdot m}$ . Therefore,

$$\begin{aligned} \mathbb{E}[m^*] &\leq \sum_{r=\sqrt{|future(x, G_{s_m}^{\text{oracle}^u})|}+1}^{\infty} \sum_{m=\sqrt{|future(x, G_{s_m}^{\text{oracle}^u})|}+1}^{r-1} C_5 \cdot e^{-D_5 \cdot m} = \\ &\sum_{m=\sqrt{|future(x, G_{s_m}^{\text{oracle}^u})|}+1}^{\infty} \sum_{r=m+1}^{\infty} C_5 \cdot e^{-D_5 \cdot m} = \sum_{m=\sqrt{|future(x, G_{s_m}^{\text{oracle}^u})|}+1}^{\infty} C_6 \cdot e^{-D_6 \cdot m} \leq \\ &C_7 \cdot e^{-D_7 \cdot \sqrt{|future(x, G_{s_m}^{\text{oracle}^u})|}}. \end{aligned}$$

The expected waiting time for  $z_{m^*}$  is the last term divided by  $\alpha \cdot \lambda$ .

<sup>29</sup>Note that  $\mathbb{E}[m^*]$  is determined by the events up to time  $t$ : take the expected value of the expression in the previous proof, where the distribution over the values of  $|future(x, G_{s_m}^{\text{oracle}^u})|$  (and of the  $s_m$ 's themselves) is conditioned on  $|future(x, G_t^{\text{oracle}^u})|$  (for the  $\text{oracle}^u$  which maximizes the expected value).



- $|future_h(x, G_t^v)| \leq n_x$
- $|anticone_h(x, G_t^{oracle})| \leq gap(x, G) + n_j =: j$

Conditioned on these relations, by Corollary 15 the event  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$  occurs w.p.  $\geq 1 - f_{post\_mine}(n_x(G_t^v), g(G_t^v), l(G_t^v))$ . All in all, conditioned on  $\mathcal{E}_t^v(x, y, \epsilon)$ , the event  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$  occurs w.p.  $\geq 1 - \epsilon$ .

**Part II:** We proceed to show that, conditioned on  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$  and on the above relations, the value of  $Risk(G_s^u, x, y)$  goes (almost surely) to 0 as time develops, for all  $u \in honest$ .<sup>30</sup>

That  $f_{pre\_mine}(l(G_s^u)) + f_{pre\_pub}(n_j(G_s^u)) + f_{post\_pub}(|future(x, G_s^u)|)$  goes to 0 as  $s$  grows follows immediately from Lemmas 24, 31, and 29. Let  $\epsilon_0 > 0$ . We now prove that after some  $\tau$  of finite expectation,  $f_{post\_mine}(n_x(G_s^u), g(G_s^u), l(G_s^u)) < \epsilon_0$ .

We claim that

$$M(oracle^u, s) + g(oracle^u, s) - n_x(oracle^u, s) \geq -2 \cdot \left| G_{[time(x), t]}^{oracle} \right| - m^* \quad (73)$$

where  $m^*$  is the variable described in Lemma 16. Assume first that  $attacker \cap G_s^{oracle} \subseteq G_s^{oracle^u}$ . Let us decompose  $future(x, G_s^{oracle^u})$  as follows:

- Blocks in  $G_{[time(x), t]}^{oracle}$ . Clearly, the number of blocks in this set does not grow with  $s$ . Their contribution is lower bounded by  $-2 \cdot \left| G_{[time(x), t]}^{oracle} \right|$ .
- Blocks in  $V_{x < y}(G_s^{oracle^u}) \setminus G_t^{oracle}$ : Every  $z$  in this set adds (+1) to  $g(oracle^u, s)$ . As  $z$  cannot decrement the value of  $M(oracle^u, s) - n_x(oracle^u, s)$  by more than 1, the contribution of this set is at least 0.<sup>31</sup>
- Blocks in  $G_s^{oracle^u} \setminus (V_{x < y}(G_s^{oracle^u}) \cup G_t^{oracle})$ : Lemma 16 guarantees that, conditioned on the event  $\widehat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$ , at least  $\left| future_a(x, G_s^{oracle^u}) \cap G_{[t, s]}^{oracle} \right| - m^*$  of the blocks that are published after some  $\tau$  and that do not belong to  $V_{x < y}(G_s^{oracle^u})$  – hence that add (-1) to  $g(oracle^u, s)$ <sup>32</sup> – add (+1) to the value of  $M(oracle^u, s)$ . In other words, at most  $m^*$  blocks from the set  $future_a(x, G_s^{oracle^u}) \cap G_{[t, s]}^{oracle} \setminus V_{x < y}(G_s^{oracle^u})$  add (-1) to  $g(oracle^u, s)$  and are not canceled out by a (+1) increment to the value of  $M(oracle^u, s)$ . The contribution of this set is therefore lower bounded by  $-m^*$ .

**Part III:** We now claim that

$$M(G_s^u) + g(G_s^u) - n_x(G_s^u) \geq -2 \cdot \left| G_{[time(x), t]}^{oracle} \right| - m^* \quad (74)$$

<sup>30</sup>In fact, we need to show that  $\max\{Risk(G_s^u, x, y)\}$  goes to 0. However, since our analysis below takes the worst case regarding  $u$ , namely, that messages from it and to it arrive at a delay of precisely  $d$ , these events are equivalent in the worst case, and thus we will relate to  $u$  as a fixed honest node.

<sup>31</sup>In fact, by the conditioning on the relation  $|future_h(x, G_t^v)| \leq n_x$ , we know that all honest blocks belong to this category, hence we can arrive at a tighter bound:  $M(oracle^u, s) + g(oracle^u, s) - n_x(oracle^u, s) \geq -\left| G_{[time(x), t]}^{oracle} \cap attacker \right| - m^*$ .

<sup>32</sup>They cannot add 0 since only strong voters are counted into these variables.

Indeed, let  $C(z)$  be the contribution of  $z$  to (73) and let  $c(z)$  be its contribution to (74). First,  $C(z) \geq -2$ , hence the contribution of all  $z \in G_{[time(x),t]}^{oracle}$  is at least  $-2 \cdot \left| G_{[time(x),t]}^{oracle} \right|$ , as previously.

Assume that  $z \in G_s^{oracle^u} \setminus G_t^{oracle}$  and that it votes  $x \prec y$ . Then  $z$  is not counted into  $M(oracle^u, s)$ , hence its contribution to  $M(oracle^u, s) + g(oracle^u, s) - n_x(oracle^u, s)$  is  $0 + 1 - 1 = 0$ , i.e.,  $c(z) = 0$ . And for the same argument  $C(z) = 0$ .

Assume that  $z \in G_s^{oracle^u} \setminus G_t^{oracle}$  and that it votes  $y \prec x$ . Then  $z \in attacker$  (by the conditioning on  $\hat{\mathcal{E}}_{t \rightarrow \infty}^{all}(x, y)$ ). In the analysis of Lemma 16 we assumed the following worst case: that for any three blocks  $v, z, w \in G_s^{oracle^u}$ , such that  $v, z \in attacker$  and  $w \in honest$ ,  $v$  votes strongly for  $z \prec w$ .<sup>33</sup> Under this worst case assumption regarding the votes of attacker blocks,  $dist\_gap(z, G_s^{oracle^u}) \leq dist\_gap(z, G_s^u)$ , as  $G_s^{oracle^u} \setminus G_s^u$  contains only attacker blocks. Thus, if  $z$  was counted in  $M(oracle^u, s)$  then it is counted also in  $M(G_s^u)$ ; in particular,  $C(z) \geq c(z)$ .

Consequently, using the analysis from Lemma 16,

$$\begin{aligned}
-m^* &\leq \sum_{z \in G_s^{oracle^u} \setminus G_t^{oracle}} c(z) \leq \\
&\sum_{z \in G_s^{oracle^u} \setminus (G_t^{oracle} \cup V_{x \prec y}(G_s^{oracle^u}))} c(z) = \\
&\sum_{z \in G_s^u \setminus (G_t^{oracle} \cup V_{x \prec y}(G_s^{oracle^u}))} c(z) + \sum_{z \in G_s^{oracle^u} \setminus (G_s^u \cup V_{x \prec y}(G_s^{oracle^u}))} c(z) \leq \\
&\sum_{z \in G_s^u \setminus (G_t^{oracle} \cup V_{x \prec y}(G_s^{oracle^u}))} c(z) \leq \sum_{z \in G_s^u \setminus (G_t^{oracle} \cup V_{x \prec y}(G_s^{oracle^u}))} C(z).
\end{aligned}$$

All in all,  $M(G_s^u) + g(G_s^u) - n_x(G_s^u) \geq -2 \cdot \left| G_{[time(x),t]}^{oracle} \right| - m^*$ .

**Part IV:** In the remainder of the proof we occasionally abbreviate  $n_x(G_s^u)$  and write simply  $n_x$ , and similarly for the rest of the variables, for convenience. Lemmas 24 and 31 imply further that there exist constants  $a, b$ , and  $W$  such that  $\Pr(k + l + 2 \cdot h + j > W) \leq e^{-a \cdot W + b}$  (as in the proof of the previous lemma, but not necessarily with the same constants). Take  $W$  such that  $e^{-a \cdot W + b} < \epsilon_0/4$ . Thus, with

<sup>33</sup>Indeed, therein we only counted honest voters in favour of honest blocks. This could be formalized using pseudo-votes, as in Lemma 19.

probability  $\geq 1 - \epsilon_0/4$ :

$$f_{post.mine}(n_x(G_s^u), g(G_s^u), l(G_s^u)) = \sum_{k=0}^{\infty} \mathcal{P}_{oiss}(3 \cdot d \cdot (1 - \alpha) \cdot \lambda, k) \cdot \sum_{h=0}^{\infty} \mathcal{P}_{oiss}(d \cdot (1 - \alpha) \cdot \lambda, h). \quad (75)$$

$$\left( \sum_{m'=M}^{\infty} \binom{n_x + j + h + m' - 1}{m'} \cdot (1 - \alpha)^{n_x + j + h} \cdot \alpha^{m'} \right)^{-1}. \quad (76)$$

$$\sum_{m=M}^{\infty} \binom{n_x + j + h + m - 1}{m} \cdot (1 - \alpha)^{n_x + j + h} \cdot \alpha^m \cdot \left( \frac{\alpha}{1 - \alpha} \right)^{(g - 2 \cdot h - k - j - l - (m - M))^+}. \quad (77)$$

For large enough  $n_x$ 's, this term is at most  $\epsilon_0/4$  away from

$$\left( \frac{\alpha}{1 - \alpha} \right)^{g + M - n_x - W} \cdot \left( \sum_{m'=M}^{\infty} \binom{n_x + m' - 1}{m'} \cdot (1 - \alpha)^{n_x} \cdot \alpha^{m'} \right)^{-1}. \quad (78)$$

$$\sum_{m=M}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m \cdot \left( \frac{\alpha}{1 - \alpha} \right)^{(n_x - m)^+}. \quad (79)$$

**Part V:** As for the first multiplicand of (78), by *Part II* of this proof, after some  $\tau$  of finite expectation:  $M(G_s^u) + g(G_s^u) - n_x(G_s^u) \geq -|G_{[time(x), t]}^{oracle}| - m^* =: D_2$  (a constant determined by time  $\tau$ ). Assume  $s \geq \tau$ . We conclude that the term  $\left( \frac{\alpha}{1 - \alpha} \right)^{g + M - n_x - W}$  is upper bounded by  $e^{D_3 \cdot D_4}$  (with  $D_3 = \ln\left(\frac{1 - \alpha}{\alpha}\right)$ ). Thus, in order to show that (78) vanishes suffice it to show that

$$\left( \sum_{m'=M}^{\infty} \binom{n_x + m' - 1}{m'} \cdot (1 - \alpha)^{n_x} \cdot \alpha^{m'} \right)^{-1}. \quad (80)$$

$$\sum_{m=M}^{\infty} \binom{n_x + m - 1}{m} \cdot (1 - \alpha)^{n_x} \cdot \alpha^m \cdot \left( \frac{\alpha}{1 - \alpha} \right)^{(n_x - m)^+} \quad (81)$$

vanishes.

The last term equals

$$\left( \Pr_{m \sim Z(1 - \alpha, n_x)}(m \geq M) \right)^{-1} \cdot \left( \Pr_{m \sim Z(\alpha, n_x)}(m \leq n_x) + \Pr_{m \sim Z(1 - \alpha, n_x)}(m \geq n_x) \right). \quad (82)$$

For large enough  $n_x$ 's, a variable distributed according to  $Z(1 - \alpha, n_x)$  converges to a normal variable with mean  $n_x \cdot \frac{\alpha}{1 - \alpha}$  and variance  $n \cdot \frac{\alpha}{(1 - \alpha)^2}$ . The last term is therefore

at most  $\epsilon_0/4$  away from

$$\left( \Pr_{z \sim \mathcal{N}(0,1)} \left( z \geq \frac{M - \frac{\alpha}{1-\alpha} \cdot n_x}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot n_x}} \right) \right)^{-1}. \quad (83)$$

$$\left( \Pr_{z \sim \mathcal{N}(0,1)} \left( z \leq \frac{n_x - \frac{1-\alpha}{\alpha} \cdot n_x}{\sqrt{\frac{1-\alpha}{\alpha^2} \cdot n_x}} \right) + \Pr_{z \sim \mathcal{N}(0,1)} \left( z \geq \frac{n_x - \frac{\alpha}{1-\alpha} \cdot n_x}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot n_x}} \right) \right) = \quad (84)$$

$$\left( \Pr_{z \sim \mathcal{N}(0,1)} \left( z \geq \frac{M - \frac{\alpha}{1-\alpha} \cdot n_x}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot n_x}} \right) \right)^{-1}. \quad (85)$$

$$\left( \Pr_{z \sim \mathcal{N}(0,1)} \left( z \geq \frac{\frac{1-\alpha}{\alpha} \cdot n_x - n_x}{\sqrt{\frac{1-\alpha}{\alpha^2} \cdot n_x}} \right) + \Pr_{z \sim \mathcal{N}(0,1)} \left( z \geq \frac{n_x - \frac{\alpha}{1-\alpha} \cdot n_x}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot n_x}} \right) \right). \quad (86)$$

We use the following inequalities due to Komatu (1955), for  $x \geq 0$ , and a standard normal variable  $z \sim \mathcal{N}(0, 1)$ :  $\frac{1}{\sqrt{2\pi}} \cdot \frac{2 \cdot e^{-x^2/2}}{x + \sqrt{4+x^2}} \leq \Pr(z > x) \leq \frac{1}{\sqrt{2\pi}} \cdot \frac{2 \cdot e^{-x^2/2}}{x + \sqrt{2+x^2}}$ .

Put  $x_1 := \frac{M - \frac{\alpha}{1-\alpha} \cdot n_x}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot n_x}}$ ,  $x_2 := \frac{\frac{1-\alpha}{\alpha} \cdot n_x - n_x}{\sqrt{\frac{1-\alpha}{\alpha^2} \cdot n_x}}$ , and  $x_3 := \frac{n_x - \frac{\alpha}{1-\alpha} \cdot n_x}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot n_x}}$ .

We obtain an upper bound on (86):

$$\sqrt{\pi/2} \cdot (x_1 + \sqrt{4+x_1^2}) \cdot e^{x_1^2/2} \cdot \left( \frac{1}{\sqrt{\pi/2}} \cdot \frac{e^{-x_2^2/2}}{x_2} + \frac{1}{\sqrt{\pi/2}} \cdot \frac{e^{-x_3^2/2}}{x_3} \right) = \quad (87)$$

$$(x_1 + \sqrt{4+x_1^2}) \cdot e^{x_1^2/2} \cdot \left( \frac{e^{-x_2^2/2}}{x_2} + \frac{e^{-x_3^2/2}}{x_3} \right) \quad (88)$$

We further observe that, for large  $n_x$ 's:  $x_2 \geq C_2 \cdot \sqrt{n_x}$  and  $x_3 \geq C_3 \cdot \sqrt{n_x}$ , for some positive constants  $C_i$  (this applies to all constants below as well). Therefore,  $\frac{(x_1 + \sqrt{4+x_1^2})}{\min\{x_2, x_3\}} \leq C_1 / \max\{C_2, C_3\} =: D_1$ . The above term is therefore upper bounded, up to a multiplicative factor of  $D_1$ , by

$$\begin{aligned} & e^{x_1^2/2 - x_2^2/2} + e^{x_1^2/2 - x_3^2/2} = \\ & e^{0.5 \cdot \left( \left( \frac{M - \frac{\alpha}{1-\alpha} \cdot n_x}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot n_x}} \right)^2 - \left( \frac{\frac{1-\alpha}{\alpha} \cdot n_x - n_x}{\sqrt{\frac{1-\alpha}{\alpha^2} \cdot n_x}} \right)^2 \right)} + e^{0.5 \cdot \left( \left( \frac{M - \frac{\alpha}{1-\alpha} \cdot n_x}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot n_x}} \right)^2 - \left( \frac{n_x - \frac{\alpha}{1-\alpha} \cdot n_x}{\sqrt{\frac{\alpha}{(1-\alpha)^2} \cdot n_x}} \right)^2 \right)} \leq \\ & e^{0.5 \cdot \left( \frac{(1-\alpha)^2}{\alpha \cdot n_x} \cdot (M - \frac{\alpha}{1-\alpha} \cdot n_x)^2 - \frac{(1-2\alpha)^2}{1-\alpha} \cdot n_x \right)} + e^{0.5 \cdot \left( \frac{(1-\alpha)^2}{\alpha \cdot n_x} \cdot (M - \frac{\alpha}{1-\alpha} \cdot n_x)^2 - \frac{(1-2\alpha)^2}{\alpha} \cdot n \right)}. \quad (89) \end{aligned}$$

Conditioned on the relation  $|future_h(x, G_t^{oracle})| \leq n_x$ ,  $M \leq future_a(x, G_s^u)$ , hence its expected value is at most  $\frac{\alpha}{1-\alpha} \cdot n_x$ . For any  $\delta > 0$ , by the Strong Law of Large Numbers, after some  $\tau$  (of finite expectation),  $\forall s \geq \tau : M \leq (1 + \delta) \cdot \mathbb{E}[M] \leq (1 + \delta) \cdot \frac{\alpha}{1-\alpha} \cdot n_x$ .

Consequently, (89) is upper bounded by

$$e^{0.5 \cdot \frac{(1-\alpha)^2}{\alpha \cdot n_x} \cdot (M - \frac{\alpha}{1-\alpha} \cdot n_x)^2 - 0.5 \cdot \frac{(1-2\alpha)^2}{1-\alpha} \cdot n_x} + e^{0.5 \cdot \frac{(1-\alpha)^2}{\alpha \cdot n_x} \cdot (M - \frac{\alpha}{1-\alpha} \cdot n_x)^2 - 0.5 \cdot \frac{(1-2\alpha)^2}{\alpha} \cdot n_x} \leq \quad (90)$$

$$e^{0.5 \cdot \frac{(1-\alpha)^2}{\alpha \cdot n_x} \cdot (\delta \cdot \frac{\alpha}{1-\alpha} \cdot n_x)^2 - 0.5 \cdot \frac{(1-2\alpha)^2}{1-\alpha} \cdot n_x} + e^{0.5 \cdot \frac{(1-\alpha)^2}{\alpha \cdot n_x} \cdot (\delta \cdot \frac{\alpha}{1-\alpha} \cdot n_x)^2 - 0.5 \cdot \frac{(1-2\alpha)^2}{\alpha} \cdot n_x} \leq \quad (91)$$

$$e^{R_1/n_x - R_2 \cdot n_x} + e^{R^3/n_x - R_4 \cdot n_x} \leq e^{-R_5 \cdot n_x}, \quad (92)$$

for some positive constants  $R_i$ , where the last inequality holds for large enough  $n_x$ 's, and the preceding inequality holds for small enough  $\delta$ 's ( $\delta < 1/n_x$ ).

Taking  $n_x$  to be greater than  $n_x > \ln(4 \cdot D_1/\epsilon_0)/R_5$  we conclude that for some large enough  $n_x$ :

$$f_{post\_mine}(n_x(G_s^u), g(G_s^u), l(G_s^u)) < 4 \cdot \epsilon_0/4 = \epsilon_0. \quad (93)$$

(Note that the expected waiting time for the first  $\tau$  such that  $\forall j \in honest : n_x(u, \psi)$  is at least some  $n_0$  is at most  $n_0 \cdot ((1-\alpha) \cdot \lambda)^{-1} + d$ : it is  $1/((1-\alpha) \cdot \lambda)$  for the creation of every honest block, and  $d$  for the last one to arrive at all nodes.)  $\square$

## 10.4 Proof of Weak Liveness (blocks)

We've seen that the error functions  $f_{pre\_mine}(l(G_s^u))$ ,  $f_{pre\_pub}(n_j(G_s^u))$ , and  $f_{post\_pub}(|future(x, G_s^u)|)$  go to 0 as  $s$  grows. For any  $s < \psi$ ,  $y \notin G_s^{pub}$ , hence by line 5 of Algorithm 3,  $g(G_s^u) = |future(x, G)| = n_x(G_s^u)$ , and  $M(G_s^u) = 0$ . In particular, the relation (73) is satisfied trivially, and the analysis in the proof of Lemma 17 applies, proving that the term  $f_{post\_mine}$  vanishes as time grows. In particular, since these functions decrease exponentially, it becomes smaller than  $\epsilon$  after a number of honest blocks in the order of  $\mathcal{O}(\ln(1/\epsilon))$  are created, and the expected waiting time for this is obtained by dividing this number by  $(1-\alpha) \cdot \lambda$  (and adding  $d$  for all honest blocks to receive these blocks).

## 10.5 Proof of Progress (blocks)

This follows immediately from the proof of Lemma 17, in which it was shown that, conditioned on the event  $\widehat{\mathcal{E}}_{\infty}^{all} t(x, y)$ ,  $f_{pre\_mine}(l(G_s^v)) + f_{pre\_pub}(n_j(G_s^u)) + f_{post\_pub}(|future(x, G_t^u)|) + f_{post\_mine}(n_x(G_s^u), g(G_s^u), l(G_s^u))$  vanishes as  $s$  grows indefinitely. In Lemma 12 it was shown that, up to a probability of  $\epsilon$ , the event  $\mathcal{E}_t^v(x, y, \epsilon)$  is contained in  $\widehat{\mathcal{E}}_{\infty}^{all} t(x, y)$  (i.e., when the former is intersected with an event of probability  $\geq 1 - \epsilon$ ).

## 10.6 Proof of Safety

**Part I:** Denote by  $risk_{acc}(G_s^u, tx, subG)$  ( $risk_{rej}$ ) the output of Algorithm 4 (respectively, Algorithm 5) when given the inputs  $G_s^u$  (for some honest  $u$ ),  $tx$ , and  $subG$  (such that  $subG$  is the past of some (possibly virtual) block). For any  $z \in [tx] \cap subG$ , denote by  $risk_{acc}^z(G_s^u, tx, subG)$  the value of the  $risk$  variable as

the loop in line 2 of *RiskTxAccept* terminates its run over  $z$ . Denote similarly  $\text{minrisk}_{rej}^{z'}(G_s^u, tx, subG)$  w.r.t. the variable  $\text{minrisk}$  in *RiskTxReject*.

We claim that, with probability  $> 1 - \text{risk}_{acc}(G_t^v, tx, subG)$ , there exists a  $\tau_{acc}$  of finite expectation such that for all  $s \geq \tau_{acc}$ , for all  $u \in \text{honest}$ , and for all  $subG' \supseteq subG$ :

$$\text{risk}_{acc}(G_t^v, tx, subG) \geq \text{risk}_{acc}(G_s^u, tx, subG') \quad (94)$$

Similarly, we claim that, with probability  $> 1 - \text{risk}_{rej}(G_t^v, tx, subG)$ , there exists a  $\tau_{rej}$  of finite expectation such that for all  $s \geq \tau_{rej}$ , for all  $u \in \text{honest}$ :

$$\text{risk}_{rej}(G_t^v, tx, subG) \geq \text{risk}_{rej}(G_s^u, tx, subG). \quad (95)$$

Assume we have proved this for all  $subG$  of size  $< k$ . We now prove this for  $subG_k$  of size  $k$ .

By the definition of  $\text{risk}_{acc}$ , there exists a  $z_{tx} \in subG_k \cap [tx]$  such that  $\text{risk}_{acc}(G_t^v, tx, subG') = \text{risk}_{acc}^{z_{tx}}(G_t^v, tx, subG')$ .

**Part II:** Denote by  $Z_2$  the set of instantiations of the third-loop-variable  $z_2$ , inside the iteration of the first-loop with  $z_1 = z_{tx}$ . By Propositions 6 and 7,  $\forall z_2 \in Z_2$ , with probability  $\geq 1 - \text{Risk}(G_t^v, (\text{vote}(z'))_{z' \in C}, z_1, z_2)$ , for any  $\epsilon'$ , after some  $\tau$  (of finite expectation),  $\forall z_2' \in (G_s^u \setminus G_t^v) \cup \{z_2\} : \text{Risk}(G_s^u, (\text{vote}(z))_{z \in C}, z_{tx}, z_2') \leq \epsilon'$ . Moreover, in the proof of Proposition 17 it was shown that the minimal  $\epsilon'$  for which this property holds at time  $s$  decreases exponentially with  $n$  (which grows linearly with  $s$ ). Thus, for all  $s$  greater than some  $\tau$ ,  $\sum_{z_2' \in (G_s^u \setminus G_t^v) \cup Z_2} : \text{Risk}(G_s^u, (\text{vote}(z))_{z \in C}, z_{tx}, z_2') \leq \sum_{z_2' \in Z_2} \text{Risk}(G_t^v, (\text{vote}(z))_{z \in C}, z_{tx}, z_2')$ .

**Part III:** Similarly, by Proposition 6, with probability of at least  $\text{Risk}(G_t^v, (\text{vote}(z))_{z \in C}, z_{tx}, \emptyset)$ , after some  $\tau$  (of finite expectation),  $\text{Risk}(G_s^u, (\text{vote}(z))_{z \in C}, z_{tx}, \emptyset) \leq \text{Risk}(G_t^v, (\text{vote}(z))_{z \in C}, z_{tx}, \emptyset)$ .

**Part IV:** Let  $\epsilon_i(G_s^u, tx, subG)$  be the series of values returned by the call to *RiskTxAccept* in line 7 of *RiskTxAccept* (when given the inputs  $(G_s^u, tx, subG)$ ) and to *RiskTxReject* in line 9 of *RiskTxAccept* (with these inputs). By the induction hypothesis, with probability  $\geq 1 - \epsilon_i$ , after some time  $\tau$ ,  $\epsilon_i(G_s^u, tx, \text{past}(z_{tx})) \leq \epsilon_i(G_t^v, tx, \text{past}(z_{tx}))$ .<sup>34</sup>

**Part V:** The above arguments show that, with probability  $\geq 1 - \text{risk}_{acc}^{z_{tx}}(G_t^v, tx, subG)$ , the sum of increments to the value of  $\text{risk}_{acc}^{z_{tx}}(G_s^u, tx, subG')$  is upper bounded by the the sum of increments to the value of  $\text{risk}_{acc}^{z_{tx}}(G_t^v, tx, subG)$ , for all  $s \geq \tau$ , where  $\tau$  is of finite expectation. As  $\text{risk}_{acc}(G_s^u, tx, subG') \leq \text{risk}_{acc}^{z_{tx}}(G_s^u, tx, subG')$ , and as  $\text{risk}_{acc}(G_t^v, tx, subG) = \text{risk}_{acc}^{z_{tx}}(G_t^v, tx, subG)$ , this proves that, with probability  $\geq 1 - \text{risk}_{acc}^{z_{tx}}(G_t^v, tx, subG)$  the inequality  $\text{risk}_{acc}(G_s^u, tx, subG') \leq \text{risk}_{acc}(G_t^v, tx, subG)$  holds.

<sup>34</sup>Technically, the indexes  $i$  on both hand-sides of this inequality should be described more carefully. To save cumbersome notation, we rely on the understanding of the reader. Informally, every instantiation of the loop-variables inside *RiskTxAccept* (when given the inputs  $(G_s^u, tx, subG)$ ) is also realized by future calls of *RiskTxAccept* (when given the inputs  $(G_s^u, tx, subG')$ ). We thus compare the results of the increments in the former to those in the latter. This is also true vice versa (for  $z_1 = z_{tx}$ ): Inside the first-loop's iteration over  $z_1 = z_{tx}$ , the exact same calls to *RiskTxAccept* and *RiskTxReject* are made, because  $\text{past}(z_{tx})$  does not evolve with time.

**Part VI:** Similar arguments prove the induction step w.r.t. *RiskTxReject*. The difference in the proof is that, since  $risk_{rej}^{z_1}$  is not a sum, rather a minimum, hence we can ignore the fact that  $anticone(z_1, G_s^u)$  may grow in time and add loop-iterations that might further reduce the value of  $risk_{rej}^{z_1}$ . Note further that the induction claim, w.r.t. *RiskTxReject*, is restricted to the case  $subG' = subG$ . Hence, the fact that the set  $Z_G([tx])$  possibly grows with time is of no consequence, since the first loop-variable is chosen from  $Z_G([tx]) \cap subG$ . We thus conclude that, with probability  $\geq 1 - risk_{rej}(G_s^u, tx, subG_k)$ , there exists a  $\tau$  of finite expectation such that for all  $s \geq \tau$  and all  $u \in honest$ :  $risk_{rej}(G_s^u, tx, subG_k) \leq risk_{rej}(G_t^v, tx, subG_k)$ .

This completes the proof of the induction claim.

**Part VII:** Algorithm 6 returns ACCEPT if and only if *RiskTxAccept* returned a value smaller than  $\epsilon$ . The above claim implies that, if  $risk_{acc}(G_t^v, tx, G_t^v) < \epsilon$ , with probability  $\geq 1 - \epsilon$ , for all  $s \geq \tau$ , for some  $\tau$  of finite expectation, for all  $u \in honest$ :  $risk_{acc}(G_s^u, tx, G_s^u) < \epsilon$ . In other words, conditioned on  $\mathcal{A}_t^v(tx, \epsilon)$ , the event  $\bigcap_{u \in honest, s \in (\tau(t), \infty)} \mathcal{A}_s^u(tx, \epsilon)$  occurs with probability  $\geq 1 - \epsilon$ .

## 10.7 Proof of Liveness

Fix some  $z_1 \in Z_G([tx])$  for  $G = G_t^v$ . The condition that until  $\psi(t)$ :  $conflict(tx) \cap G_s^{pub} = \emptyset$ , implies that lines 6 and 7 of *RiskTxAccept* do not contribute to the value of  $risk_{acc}(G_s^u, tx, subG)$ . The assumption  $\sum_{[tx_i] \in inputs(tx)} RiskTxAccept(G_t^v, (vote(z))_{z \in C}, [tx_i], G_t^v) < \epsilon/2$  implies that, with probability  $\geq 1 - \epsilon/2$ , the overall contribution of the fourth loop to the value of  $risk_{acc}^{z_1}(G_s^u, [tx], G_s^u)$  is at most  $\epsilon/2$  (after some  $\tau$ ). Finally, by Proposition 8, the contribution of line 3 to  $risk_{acc}^{z_1}$  is less than  $\epsilon/2$ , after some  $\tau$  of finite expectation. We conclude that after some  $\tau$  of finite expectation, the value of  $risk_{acc}^{z_1}(G_s^u, [tx], G_s^u)$  is smaller than  $\epsilon/2 + \epsilon/2 = \epsilon$ , for all  $s \geq \tau$  and  $u \geq s$ , hence  $risk_{acc}(G_s^u, [tx], G_s^u) < \epsilon$ , which implies the event  $\bigcap_{u \in honest, s \in (\tau(t), \infty)} \mathcal{A}_s^u(tx, \epsilon)$ .

## 10.8 Proof of Progress

The proof of this proposition is similar in structure to that of Proposition 3. Therein we have already argued that the contributions to the value of  $risk_{acc}^{z_{tx}}$  (and similarly for  $risk_{rej}^{z_{tx}}$ ) of lines 3 and of 6 go to 0; and the increments of lines 7 and 9 go to 0 by the induction hypothesis. Thus,  $risk_{acc}(G_s^v, tx, G_s^v)$  goes to 0 as time develops, with probability  $\geq 1 - risk_{acc}(G_t^v, tx, G_t^v)$ . As  $\epsilon > risk_{acc}(G_t^v, tx, G_t^v)$ , we conclude that, with probability  $\geq 1 - \epsilon$  Algorithm 6 returns ACCEPT for all  $G_s^u$  with  $s \geq \tau$  and  $u \in honest$ .